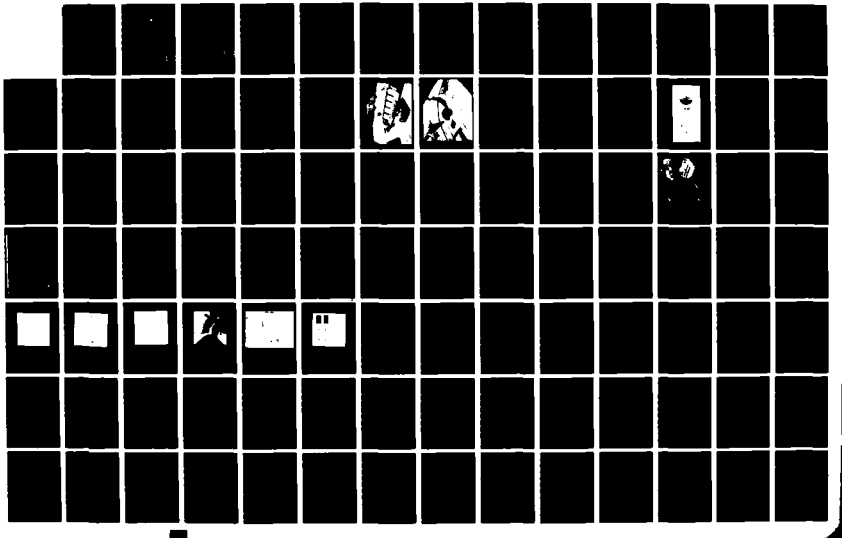
AD-A159 218 THE INTERACTIVE GENERATION OF ALPHANUMERICS AND SYMBOLOGY WITH DESIGNS ON THE FUTURE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 1/3

UNCLASSIFIED K A ADAMS JUN 85 AFIT/GCS/MA/85J-2 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

REPRODUCED AT GOVERNMENT EXPENSE

AD-A159 218

THE INTERACTIVE GENERATION OF ALPHANUMERICS AND SYMBOLOGY WITH DESIGNS ON THE FUTURE

THESIS

AFIT/GCS/MA/85J-2 Karyl A. Adams

DTIC FILE COPY

This document has been approved for public release and sale; its distribution is unlimited.

DTIC
SEP 18 1985
S
A

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/MA/85J-2

A DISPLAY ENVIRONMENT SUPPORTING THE INTERACTIVE GENERATION OF ALPHANUMERICS AND SYMBOLOGY WITH DESIGNS ON THE FUTURE

THESIS

AFIT/GCS/MA/85J-2 Karyl A. Adams

DTIC
ELECTE
SEP 18 1985
S D
A

Approved for public release; distribution unlimited

Original contains color plates; All DTIC reproductions will be in black and white.

AFIT/GCS/MA/85J-2

A DISPLAY ENVIRONMENT SUPPORTING THE INTERACTIVE GENERATION OF ALPHANUMERICS AND SYMBOLOGY WITH DESIGNS ON THE FUTURE

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Karyl A. Adams, BSEE

June 1985

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Approved for public release; distribution unlimited

DTIC COPY INSPECTED

Acknowledgments

This thesis effort has evolved from several sources. The most significant driver for the work was the many long, frustrating hours spent developing HUD symbology for Flight Dynamics Laboratory simulation programs. Typical software development to support the simulations required much hand-coding, normally adding, deleting, or modifying existing modules. The existing software structure at the time was poorly designed from the standpoint of re-useability. The majority of time was indeed spent doing trivial, almost mindless work.

A colleague of mine, Mr. Terry Christian, who had worked extensively with the available graphics system, and I discussed ways to improve our plight. Out of those discussions, the first germ of an idea to structure the software modules for true re-useability was born.

A second critical event that started my thinking toward ever more challenging goals was the time I spent in the compiler sequence with Major (then Capt) Roie Black working with implementing a subset of an Ada compiler. Major Black is an ardent advocate for making computers actually 'work' for you. His thoughts and ideas on the subject of automating massive software systems was, to say the least, inspiring and

thought provoking.

Because of my interest, and heavy involvment, in graphics development in my full-time job and my growing attraction to automated tools, the idea for DESIGNS was formulated. Major Black continued his support by becoming the advisor for my thesis effort. Throughout the time he was at AFIT, he continually provided me with challenging ideas and recommendations for the concept.

As a part-time student completing a master's degree program, the thesis often posed an interesting conflict with my day to day work. The development effort has been quite prolonged, and as a result has changed significantly over the two and a half year period. It has been an interesting maturation period, both for the concept of the thesis and myself. The final thesis product, while far from a complete automatic environment, I feel is a stronger design and set of tools than it would have been otherwise.

During this period of time there have been many individuals who, by their continuing encouragement and support, have helped me to stick with and complete my thesis. I wish to thank Major Roie Black for the inspiration of his ideas and his steady encouragement to FINISH. Special thanks also to Mr. Terry Christian and his willing ear to listen to my ideas, and help me to see when they did, and didn't, address the needs of our facility.

I would also like to thank the readers on my committee. Dr. Matthew Kabrisky and Mr. Paul E. Blatt gave of their time to provide information and comments on the thesis development. Mr. Blatt, as the branch chief of the Control Synthesis Branch, also served as the official sponsor for the effort.

There is a group of people to whom I owe a great deal in actually completing my master's program. The first is Mr. David Lair. Dave was my supervisor in the lab and supported my educational endeavors whole-heartedly. Without his support, it would have been impossible to have scheduled classes and thesis around the demands of the facility.

The second special thanks goes to Mr. Chuck Richard. He was initially a reader on my committee, and willingly stepped in as advisor when Major Black was reassigned. This required much more of his time and efforts to get caught up to speed on a development effort that was over a year old. I deeply appreciate his patience and understanding as he has guided me to completing the thesis.

Another source of strong support came from my friend and colleague, Capt Patricia Lawlis. She has encouraged when I was ready to quit, pushed when I was being lazy, and of equal importance, left me alone when I was too fed up to listen. Without her encouragement, this effort may never have been finished.

My final sincere thanks go to my family, particularly my mother and sister. They have provided constant emotional support throughout the effort. Their kind thoughts and prayers have meant much.

Without each of these people, I may never have finished what was a prolonged task. In completing the effort, I think of each of you and thank you for your support.

Karyl A. Adams

Contents

List of Figures

List of Tables

List of Acronyms

ACM	Association for Computing Machinery, Inc.
AFTI	Advanced Fighter Technology Integrator
AFWAL	Air Force Wright Aeronautical Laboratories
AMRL	Aerospace Medical Research Laboratory
ANIP	Army/Navy Instrumentation Program
ANSI	American National Standards Institute
APSE	Ada Programming Support Environment
ASCII	American Standard Code for Information Interchange
ASP	Associative Structure Package
CAD	Computer Aided Design
CADET	Computer Aided Design and Evaluation Techniques
CAM	Computer Aided Manufacturing
CIG	Computer Image Generation
CRTs	Cathode Ray Tubes
DDI	Digital to Digital Interface
DESIGNS	Display Environment Supporting the Interactive Generation of alphaNumerics and Symbology
EADI	Electronic Attitude Direction Indicator
EAI	Electronic Associates Incorporated
EHSD	Electronic Horizontal Situation Display
GKS	Graphical Kernel System

HDDs	Heads Down Displays
HOL	Higher Order Language
HUDs	Heads Up Displays
I/O	Input/Output
ISO	International Standards Organization
LEDs	Light Emitting Diodes
NALPS	North American Presentation Level Protocol Syntax
PITS	Pathway in the Sky
R & D	Research and Development
SADT	Structured Analysis and Design Technique
SDS	Symbology Development System
SEL	Systems Electronic Laboratories
SIGGRAPH	Special Interest Group in Graphics
TAACE	Tanker Avionics and Aircrew Complement Evaluation
TAWS	Total Aircrew Workload Study
TDD	Three Dimensional Drawing
TF/TA	Terrain Following/Terrain Avoidance
THREAD	Three Dimensional Editing and Drawing
TSDs	Tactical Situation Displays
VDI	Virtual Device Interface
VG	Vector General
4GLs	Fourth Generation Languages

Abstract

This ~~development effort~~ thesis investigated the available methods for implementing human-computer interfaces using sophisticated graphics systems, with the goal of designing and implementing an advanced graphics development environment. Such a developmental laboratory is necessary to support current and futuristic crew station design for display-oriented cockpits. The result of this effort was the development system - DESIGNS.

The DESIGNS software system was tailored specifically for real-time, pilot-in-the-loop, research and development simulation. Critical features of such a system include fast turn-around time, and the capability to expand as enhanced tools are developed.

The resulting system supports interactive development of heads-up and heads-down display symbology. It generates correct display formatting information for multiple target graphics devices. Future enhancements include rehosting to a more powerful workstation, preferably in Ada, and continued addition of graphics development tools supporting simulation needs.

Additional keywords: real time; human factors engineering; DESIGNS (Display Environment Supporting the Interactive Generation of Displays and Symbology)

A Display Environment Supporting the Interactive Generation of alphaNumerics and Symbology with DESIGNS on the Future

I. Introduction

Powerful computers and software techniques have made most computer users dream of 'pleasant' working environments in which the computer does much of the work. Such environments are designed with 'intelligence' in the sense that they automate many of the overhead tasks involved in the preparation of software packages. Users developing graphics software are certainly no different. They also envision computer supported environments, tailored to their tasks, which provide interactive tools for the development, modification, and testing of display formats.

Software capabilities have matured to such a point that sophisticated environments, supporting all phases of graphical development, can be postulated and then implemented on existing computer systems. The maturity of the technology is such that graphical development work can be raised to a higher level than that of direct programming and reprogramming of graphics hardware. Picture format definition, how the final version of the picture looks and moves on a display monitor, can be accomplished at a more

abstract design level with the appropriate support tools. With graphics support tools available, the designer can, and in most cases should, be removed from the numerous mundane tasks of final software coding. Then attention can be focused on the critical issues of defining those elements to be included within the picture format and testing the resultant graphical displays for adherence to the original specifications [Dudley, 1982 : 54].

Rapidly maturing graphics capabilities offer many attractive alternatives for traditional problem solutions. One area in which graphics is having a serious impact, is that of cockpit design for military aircraft. The military, in particular the Air Force, is evaluating the feasibility, economy, and effectiveness of using extensive cockpit graphics [Mulley, 1980 : 14-15; Vokits and Waruszewski, 1980 : 4-8].

Future Air Force cockpit designs will include cathode ray tubes (CRTs), flat panel, light emitting diode (LED), and other advancing technologies. The all-electronic cockpit of the future is being tested today. Figures 1 and 2 illustrate futuristic transport and fighter flight deck configurations, and depict the potential dominance of graphical displays within the aircraft environment [Wilson and Bateman, 1979 : 10].

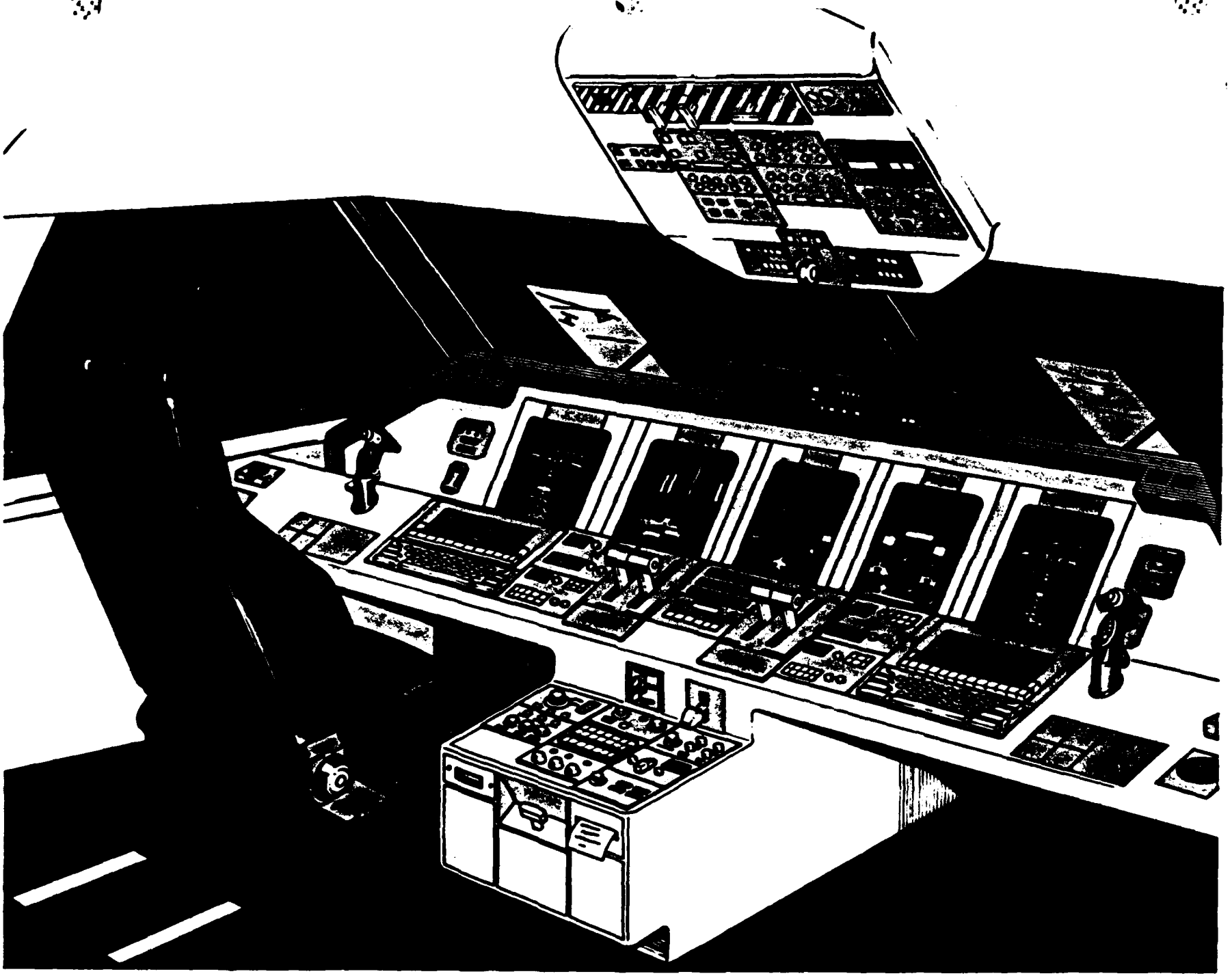


FIGURE 1 - Futuristic All-Electronic Transport Flight Deck

FIGURE 2 - Advanced Fighter Display Concepts

Displays can replace much of the current instrument panel and increase system reliability and maintainability. The inherent flexibility of displays provides a mechanism for increasing aircraft subsystem integration, thus decreasing the pilot workload and improving mission performance [Aviation Week Staff, 9 Nov 1981 : 181-188, 191-193; Bateman, 1978 : 4-10; Hare, 1978 : 17-19; Washburn and Tibor, 1979; Whitaker, 1981 : 505-510].

With the new graphics oriented technology come the inevitable questions. The hardware installed in military weapon systems and the software systems driving them require careful design to maximize their potential in the cockpit. Human use and perception of the data encoded in the display become critical [Mulley, 1980 : 15-18; Tucker, 1981 : 59-62]. Studies to determine how the hardware/software systems should interact with crew members are being conducted in simulation.

Simulation facilities are therefore expanding capabilities to include sophisticated graphics systems with which to evaluate these issues. These systems can generate complex threat, sensor, and navigation displays which are so critical to military activities. Tools to support the expanding graphics facilities and the intense software development are needed and are, in many cases, rudimentary. Intelligent graphics support environments, tailored for Air Force simulation needs, provide the capability to improve graphic development in several areas [Mysing and Gravely, 1980 : 37-38; Riesenfeld, 1978 : 115-122; Wall and others,

1980 : 19-120].

This thesis defines a complete graphics support environment developed for use in real-time flight simulation. A subset of the defined tools, focusing on the actual preparation of software for Heads-Up Displays (HUDs), is implemented. The completed product provides a highly integrated, interactive environment in which the user has access to an extensive toolset, tailored for simulation needs. From the toolset the user selects those capabilities necessary to accomplish any phase of the graphics development task at hand. The capabilities and tools discussed here are an outgrowth of the phenomenal accomplishments in graphics and computer languages over the past two decades.

Historical Background

With the advent of Ivan Sutherland's doctoral dissertation, "Sketchpad", in 1963, a new avenue for human/computer communication was opened. Sutherland perceived that the added dimension of a graphics display could provide significant benefits for the user. The Sketchpad design explored, defined, and expanded this new communications device. Its use marked the beginnings of serious graphics development [Sutherland, 1963 : 2-19].

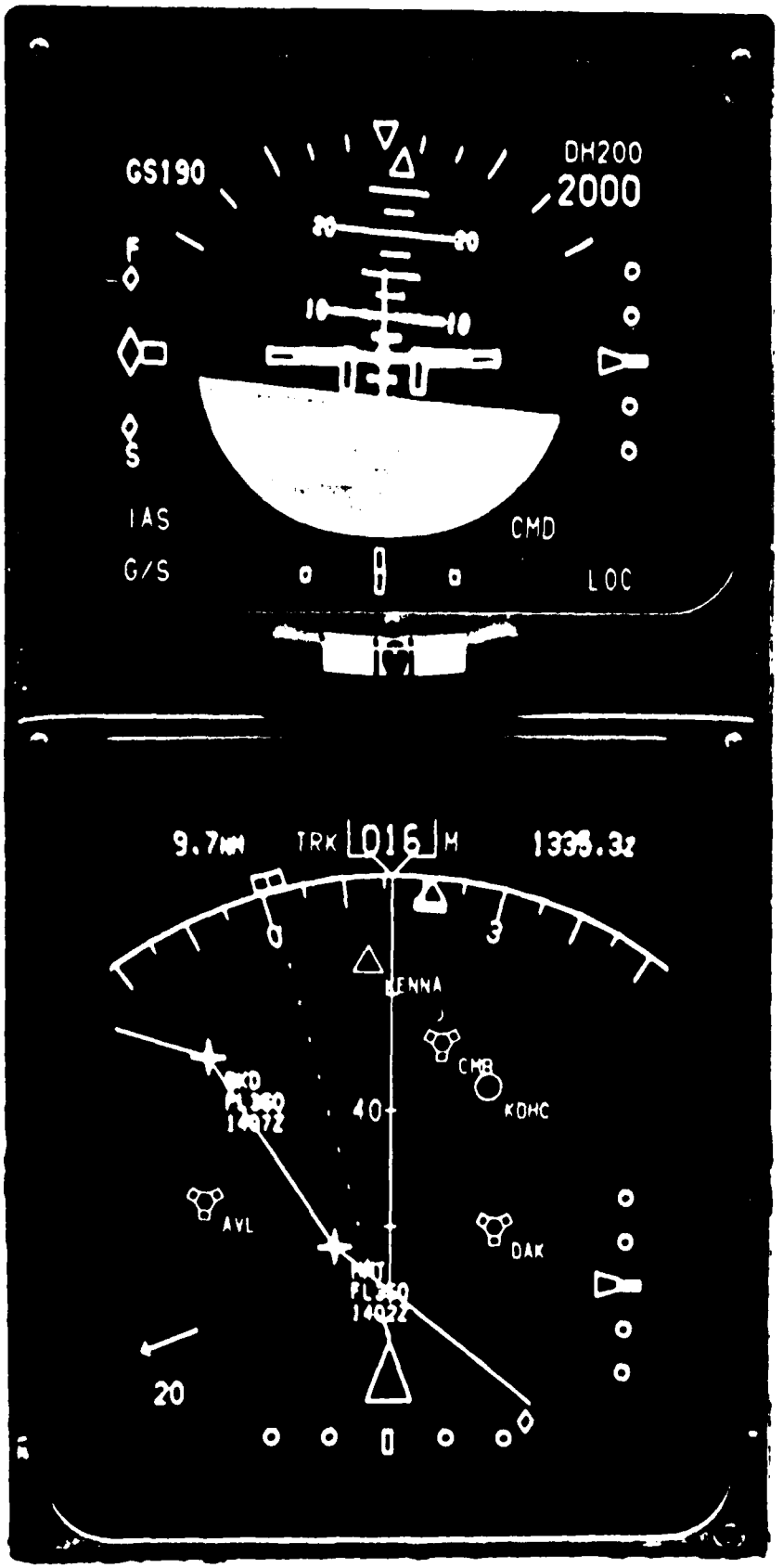
From the relatively simple line drawing techniques and cumbersome display hardware introduced with Sketchpad, advances in both software and hardware have provided

increasingly powerful graphics tools. Today, graphics use extends into numerous areas of endeavor, including games, business use, and technical support.

In the past few years graphic displays have made their way into the aircraft cockpit. Improving technology, with greater system speed and larger, faster memory, makes in-cockpit graphics a practical and economic way to display flight related information. Diversity of formats, which can be tailored to the mission activity and aircraft need, opens new horizons for data display. No longer must dedicated pieces of hardware, performing a single function, be used [Jauer and Quinn, 1982 : 1-6].

Applications of display technology in the civil aircraft sector have been extensive. Notable efforts include the Boeing 757/767 series aircraft with a complete complement of graphics units for instruments, weather radar, navigation, and system monitoring; and Britain's Royal Aircraft Establishment BAC1-11 research aircraft which is a flying testbed for high resolution, full-color, multi-function display systems. Many commercially available display units from Rockwell-Collins, Sperry Flight Systems, Bendix, Smiths Industries, and others are flying in both American and foreign airlines [Royal Aircraft Establishment, 1981 : 1-12; Smiths, 1980 : 27-33]. The Electronic Horizontal Situation Display and Electronic Attitude Direction Indicator (EHSD/EADI) symbology shown in Figure 3 is typical of color displays available and in use [FIGD, 1982].

ELECTRONIC ATTITUDE DIRECTOR INDICATOR (EADI)



ELECTRONIC HORIZONTAL SITUATION INDICATOR (EHSI)

FIGURE 3 - Representative Displays in Service Today

The military ventures to date have been more cautious [Vokits and Waruszewski, 1980 : 4]. However, as tactical missions grow in complexity, the use of computer based cockpits with graphics has increased. Both the F/A-18 and the F-16 are using some form of displays for system management information. The formats for these displays have evolved more cautiously, often strongly resembling the dedicated equipment which they replace. Indicative of this evolutionary process are the different 'displays' used in three fighter aircraft for weapons control that are shown in Figure 4 [Jauer and Quinn, 1982 : 1-2; AFTI, 1980 : 1-5]. The displays within the cockpit have successfully replaced older, existing equipment. The enhanced capabilities have been well received by crewmembers. Cockpits for future military aircraft are therefore being designed with computer graphics included as standard equipment.

Escalating costs of flight test programs require that preliminary research, design, and testing of display formats be done in ground-based simulation facilities prior to use in the actual flight environment. Displays must be evaluated to determine both their effectiveness in transferring data to flight crews and their practicality for the cockpit. Such design and testing efforts require extensive graphics support [Jauer and Quinn, 1982 : 118-119; Lizza and others, 1983 : 1-5].

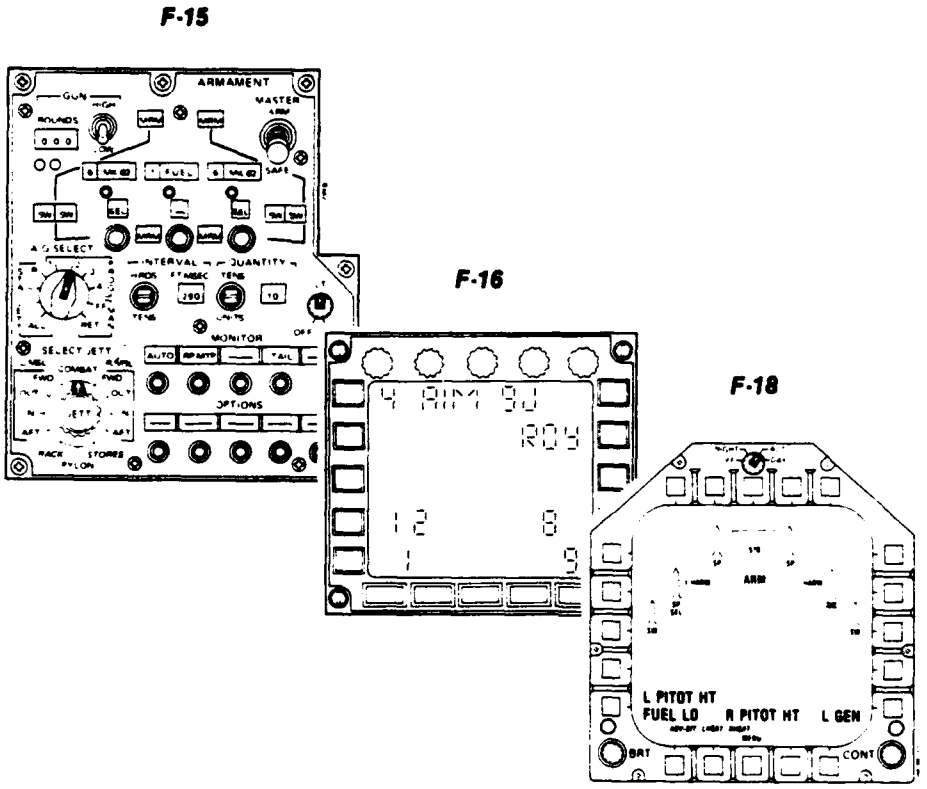


FIGURE 4
Graphic Replacements for Dedicated Control Heads -
An Evolutionary Process

The key for effective development of graphics during the simulation testing cycle depends on having graphics support tools available. The challenge of developing these tools is of major import to the simulation community if it is to provide timely support for in-cockpit graphics research at lower cost.

Scope of the Thesis

One of the most important goals of the thesis effort was to provide a more powerful, and more useable graphics environment for the personnel working the graphics area of real-time, pilot-in-the-loop, research simulation. Key to this effort has been the definition of an executive structure which serves to integrate available graphics tools, and the definition of DESIGNS - Display Environment Supporting the Interactive Generation of alphaNumerics and Symbology.

The executive provides a super-structure which loosely integrates all available graphics tools within the facility. DESIGNS is the most complex, and significant of these tools as it provides the potential for a revolutionary new way of developing real-time simulation graphics in the future.

The function of the DESIGNS environment is to assist with the development of picture formats required for real-time, ground-based flight simulation. To do this successfully, DESIGNS must provide tools which augment, automate, or eliminate some of the time consuming tasks that currently must be accomplished by hand.

The typical scenario for developing and testing graphics displays in simulation involves six major phases:

(1) determine initial requirements for cockpit display(s), that is what scenes are to be put on the display screen(s) in the cockpit,

(2) program the selected graphics system(s) to generate and update the desired display(s),

(3) have test crews fly, using the display(s) in

typical, simulated scenarios,

(4) modify the display(s) based on crew suggestions,

(5) retest the new display(s),

(6) repeat steps 4 and 5 until the desired level of crew acceptance and performance has been met.

The bulk of the work lies in step 6, 'tweaking' the display format until it is deemed acceptable. The initial display design and software development often is a modification of existing cockpit symbology, thus not a major development. There tend to be accepted sets of symbols to accomplish specified tasks. Variations on these sets, rather than dramatically new symbology sets, evolve with new testing.

This scenario, replayed for multiple simulations, soon leads to tremendous replication of effort. Much of the work is repetitive or purely computer related, requiring either little innovation or no graphics design work. Significant amounts of time can be required to modify program code and process it to run with slight variations from the preceeding configuration.

The purpose of a tool such as DESIGNS is to automate those tasks which can be, provide tools to assist in the design and eventual modification of displays, and provide a consistent interface to the variety of tools and devices the designer may have available. Thus the tools can reduce development time and permit the designer to focus on design issues rather than the mechanics of the computer and graphics

hardware.

The initial thrust of the DESIGNS effort is to provide an alternative to the manual creation of display software for simulation efforts of the type currently being undertaken. This can be accomplished by providing automated tools which perform the repetitive tasks, and support basic graphics drawing primitives. Futuristic capabilities in the cockpit lead to more complex graphics requirements, including pictorial displays - the concept of 'painting a picture'. The definition of DESIGNS supports this extension in theory, but did not address it in detail during development.

Conceptually, DESIGNS is modeled as depicted in Figure 5. The user is presented with a view of the DESIGNS capabilities (tools) through the user interface. This interface provides the only window the user has into the toolbox. Through the structure of the user interface, tools may be discerned and then accessed. No attempt is made to fully define the tools, except in terms of their behavior, to the user. Therefore the only 'tool' the user must learn to use is the interface itself. By using the tool set available, the user can design, modify, and test displays more easily and quickly.

Even further removed from the user are the actual graphics devices that may be available through the DESIGNS system. Again the user knows these devices in terms of information available through the user interface. DESIGNS provides the device dependent software (interpreter) that

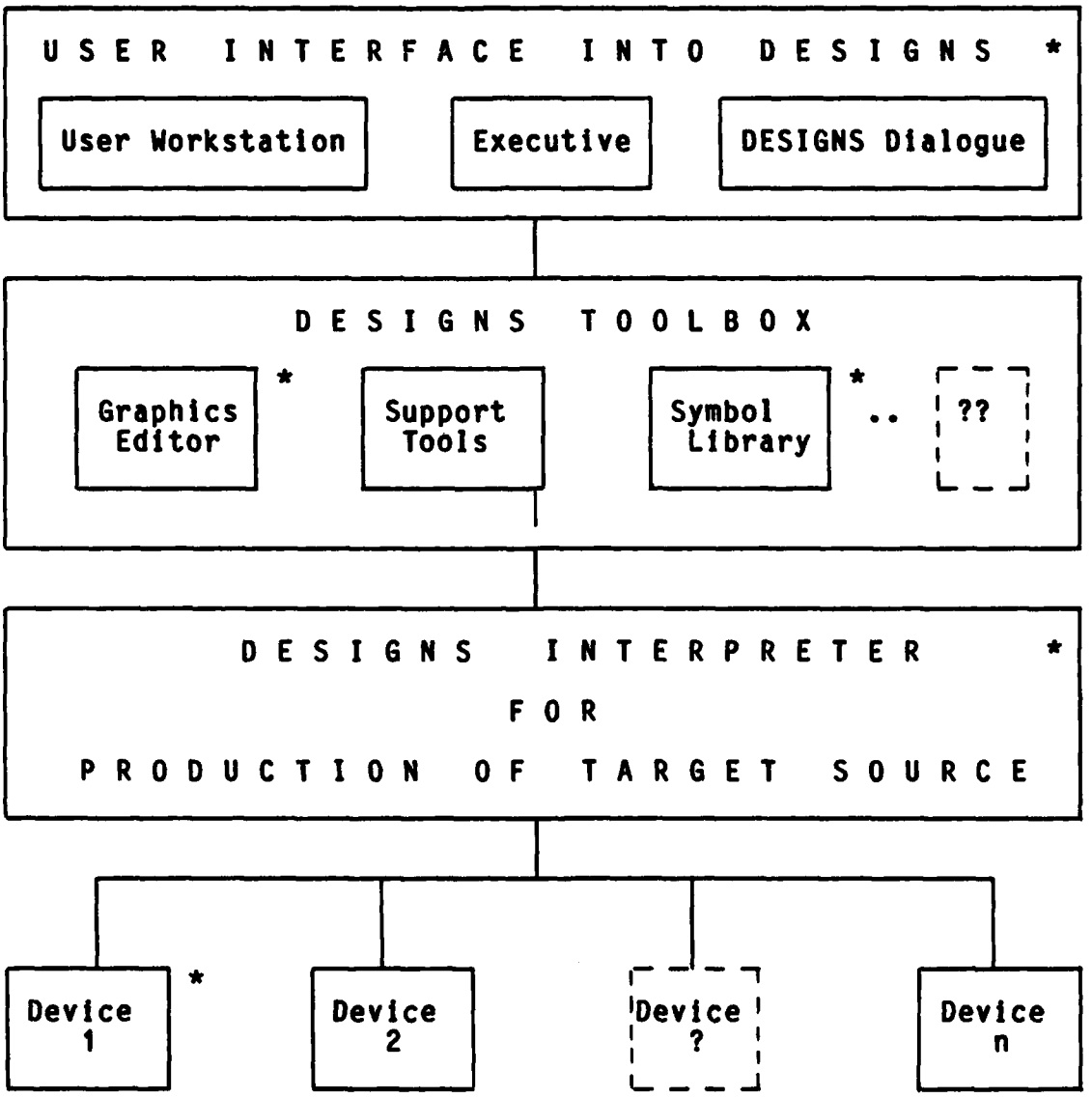


FIGURE 5
Conceptual Model of DESIGNS

translates the user's graphics design specifications into a format that a specific device can use. This portion of the DESIGNS environment may not be directly accessed by the user, as evidenced in the figure.

As indicated by the figure and briefly discussed here, the DESIGNS environment provides the user with a set of user support tools. It provides a single, defined method for using the tools, and requires no detailed knowledge of the hardware or software systems in order to work with the tools. The only portion of DESIGNS visible to a user is this user/environment interface.

The set of tools provided to the user supports the many phases of graphics work. As a minimum these tools provide aids for designing and modifying picture formats, with support to save and retrieve work that has been accomplished. The concept provides a mechanism by which other tools can be added to the toolbox as they are developed in the future.

The development and testing phases of DESIGNS served as the proof of concept. Since the DESIGNS environment was too extensive to develop in its entirety, a portion, identified as critical to the design concept, was implemented and tested. A thread through the system, which directly supports software development for HUDs, was selected as the candidate for implementation. This effort included issues of the user interface, a baseline set of graphics tools, and an interpretive mechanism for a single target device. Areas which were developed as part of the research effort are indicated in Figure 5 with an asterisk (*). This indicates the path through the system which was determined to be critical for success.

Overview of Technical Issues

Solutions which may potentially satisfy the graphics development needs will not be found by considering only those techniques available within isolated technical disciplines. Developing the DESIGNS environment definition involved issues traditionally identified with several distinct technical areas. A satisfactory solution for the design problem must be defined from an interdisciplinary perspective addressing four distinct areas.

This report is structured to discuss each of these technical issues in detail as well as to provide the necessary supporting information. The second chapter explains the importance of graphics to Air Force research and development simulation. Trends in the use of graphics in the cockpit are discussed to provide a better perspective on the problem.

The third and fourth chapters discuss problem definition and requirements. Chapter three provides the details of the problem definition and scopes the effort proposed. Chapter four defines the functional system requirements.

The actual implementation of a system having the functional capabilities specified, places a complex task in the hands of the system designer. To accomplish such a task, several technical areas, such as real-time simulation, human factors engineering, and computer systems engineering, must be carefully analyzed. If any facet of the effort is

short-changed, the resulting tool will not be accepted by the potential users. The DESIGNS system was developed based on techniques and methodologies available within each of these disciplines. The fifth and sixth chapters describe the DESIGNS environment implementation and the testing of the subset of DESIGNS tools.

The seventh, and final, chapter of the thesis presents the author's conclusions regarding the effort to date and critiques the effort with respect to the original goals. The overall success and feasibility of extending DESIGNS is discussed. Recommendations for future activities and extensions to the DESIGNS environment capabilities are made.

Additional information, which supports the project design, is presented in the appendices. This information expands on the issues raised by the system requirements and provides the basis of knowledge from which the preliminary design was defined. Such knowledge improves the reader's ability to understand the various design decisions, but is not critical to system use.

Issues specific to the simulation needs that impact this project are defined in Appendix A. Simulation questions require insight and understanding of both real-time aircraft simulation demands and cockpit design. Specific facility needs and resources strongly tailor parts of the design. These needs guide the definition of requirements and capabilities. The available assets dictate what physical devices must be supported.

A critical area that must be considered is the human factors design of the system. The importance of the user interface is discussed in Appendix B of the thesis. A complete, technically correct set of graphics tools packaged with a poor user interface is of minimal use. The needs and capabilities of the intended end user must be identified. Once identified, these points determine the user interface definition. Careful attention will be given to the efforts in the area of fourth generation languages and the impacts they have on both software and user interfaces.

The hardware and software issues pertinent to the design definition of any system are often closely coupled. It is frequently difficult to discuss one without referencing the influence of the other. This is certainly true in discussing the hardware and software technical issues which have helped to mold the DESIGNS system definition. DESIGNS is a major software design and implementation effort and as such has been influenced by existing, proven techniques in the graphics area. However, since the primary goal is to provide development support for a plethora of dissimilar target devices, the DESIGNS definition is heavily dependent on a thorough understanding of the hardware capabilities. Both current hardware capabilities and future growth potentials must be understood.

It is necessary to look at each area in turn critically, and then to discuss the integration of these closely related issues. Appendix C discusses pertinent hardware capabilities

and software techniques that affected the DESIGNS definition. Key issues concerning the hardware/software integration are identified and their relevance to the effort detailed.

The methodology of Structured Analysis and Design Technique (SADT) developed by Softech was used during the design phase of the project. The resultant SADT diagrams describing the detailed DESIGNS definition are contained in Appendix D. This set of charts defines the entire postulated environment. There is more detail contained within those charts defining the implemented subset of DESIGNS than in those defining future capabilities.

Pseudocode was used to develop the design of the software modules. This serves both as a design tool and as documentation for the system. Appendix E contains the pseudocode that was developed for the tools actually implemented.

A user's guide for the DESIGNS system is also provided. This guide, which is included as Appendix F, provides a step by step description of how to use the software. It documents those tools which have been developed and tested to date.

As discussed previously, one of the important outgrowths of the thesis was the definition and development of a high level executive program which serves to integrate all the graphics tools available within the host facility. While this executive is not the focal point of the thesis, it is an important outgrowth of it. To provide more complete references, Appendices G and H contain information specific

to the graphics executive that was developed in parallel with DESIGNS.

II. Graphics in R & D Simulation

To understand the need for the DESIGNS system, it is necessary to understand the role of graphics development and testing in simulation. Real-time, man-in-the-loop simulation is a useful tool for aircraft development and evaluation. Simulation activities are supported by, and based on the design and evaluation efforts that precede them.

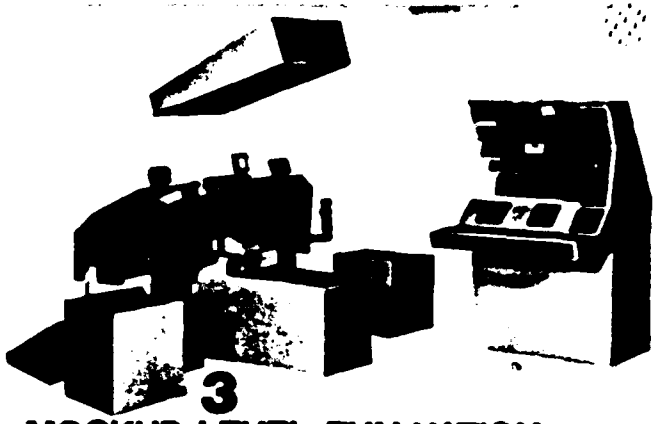
The activities that precede simulation have been carefully defined and refined over the last several years. The accepted procedure, in government and industry alike, is an iterative one consisting of analysis, design, and evaluation phases. Low-cost, mock-ups study static display formats in an attempt to lower costs of real-time simulation.

The design process used in developing a crew system design is illustrated in Figure 6. Not all steps of this process are executed for every program, but the philosophy of analysis, design, and testing are critical. There is a very real need to provide good graphical tools to support the various phases of this design process.

The increased need for graphic related software in simulation can be seen from the nature of the design process. The use of graphical representations within the aircraft cockpit is not new. The historic use of graphics in the cockpit, coupled with a brief description of the potentially demanding roles graphics may support in the near future, develops an appreciation for the DESIGNS concept.

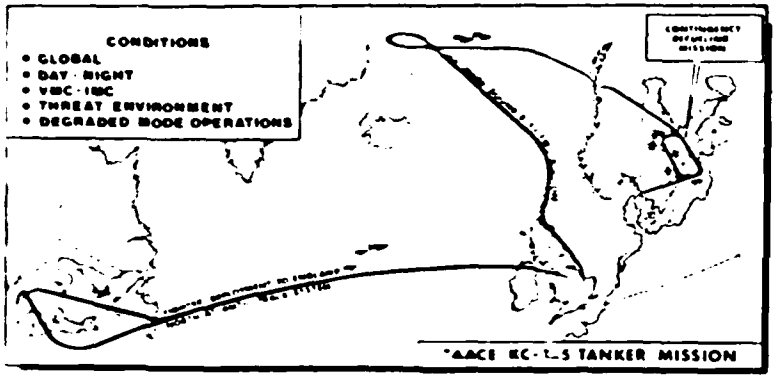
2 PRELIMINARY DESIGN

- CONTROL/DISPLAY LAYOUT
- CHECKLIST/PROCEDURES
- FUNCTION DETERMINATION



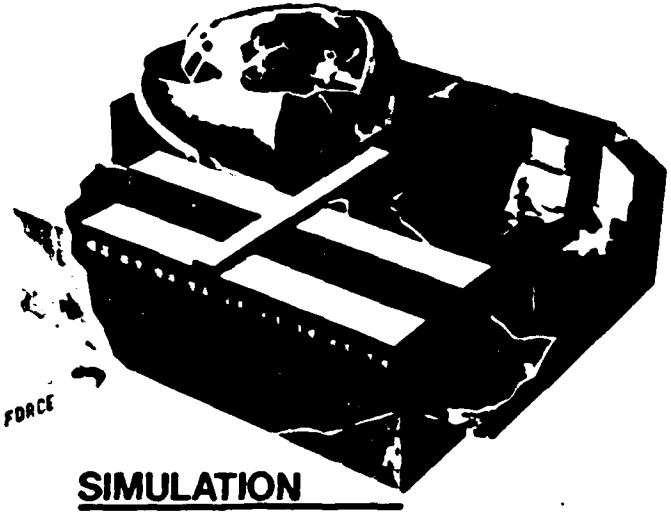
3 MOCKUP LEVEL EVALUATION

- USER PARTICIPATION
- LARGE SCALE SCREENING



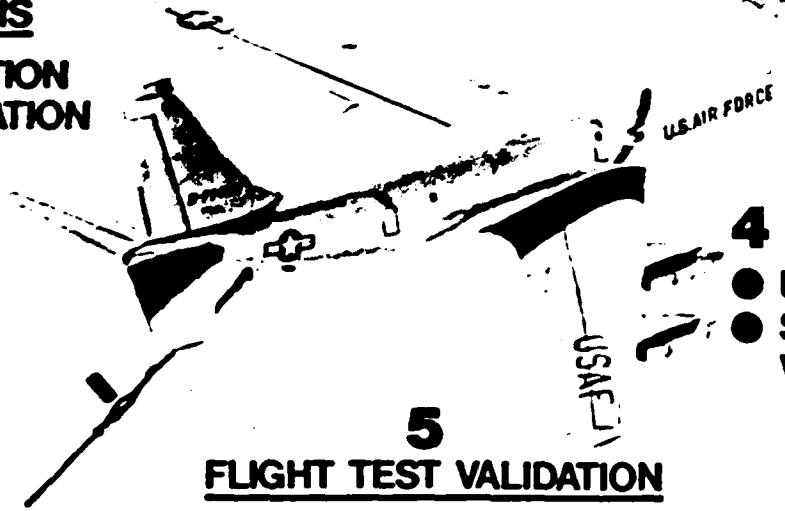
1 MISSION ANALYSIS

- INFLIGHT OBSERVATION
- SCENARIO PREPARATION



SIMULATION
4 EVALUATION/VALIDATION

- USER PARTICIPATION
- SPECIFIC CONCEPT VERIFICATION



5 FLIGHT TEST VALIDATION

CREW SYSTEM DESIGN PROCESS

FIGURE 6 - Crew System Design Process

Work in cockpit displays began in the early 1950's with the Army/Navy Instrumentation Program (ANIP). The ANIP goal of integrating the fighter cockpit was never realized primarily due to the rudimentary technology available at the time. The display system performance was notably inferior to pilot expectations and was not accepted. Strong prejudice among pilots involved in the project seriously hindered future attempts to introduce graphics as an integral portion of cockpit instrumentation [Quinn, 1982 : 1, 62].

Since the original trial efforts, much has changed with available technology and military needs. Today's airplanes have greatly improved performance capabilities with seemingly unlimited potential. They are tasked with increasingly demanding mission roles. As mission scenarios become more demanding and the aircraft more complicated to operate, the amount of data to be properly evaluated by the crew members increases accordingly.

Typical mission scenarios for large aircraft (bombers and transports) and high performance fighters are grueling. They demand periods of intense mental activity on the part of crewmembers to accomplish critical tasks. Bomber missions can require long flight hours, involve numerous rendezvous points with other aircraft, with various weapons launches as illustrated in Figure 7. The fighter mission, as depicted in Figure 8, can involve high-speed terrain following/terrain avoidance (TF/TA), threat avoidance, plus aquisition and identification of enemy targets.

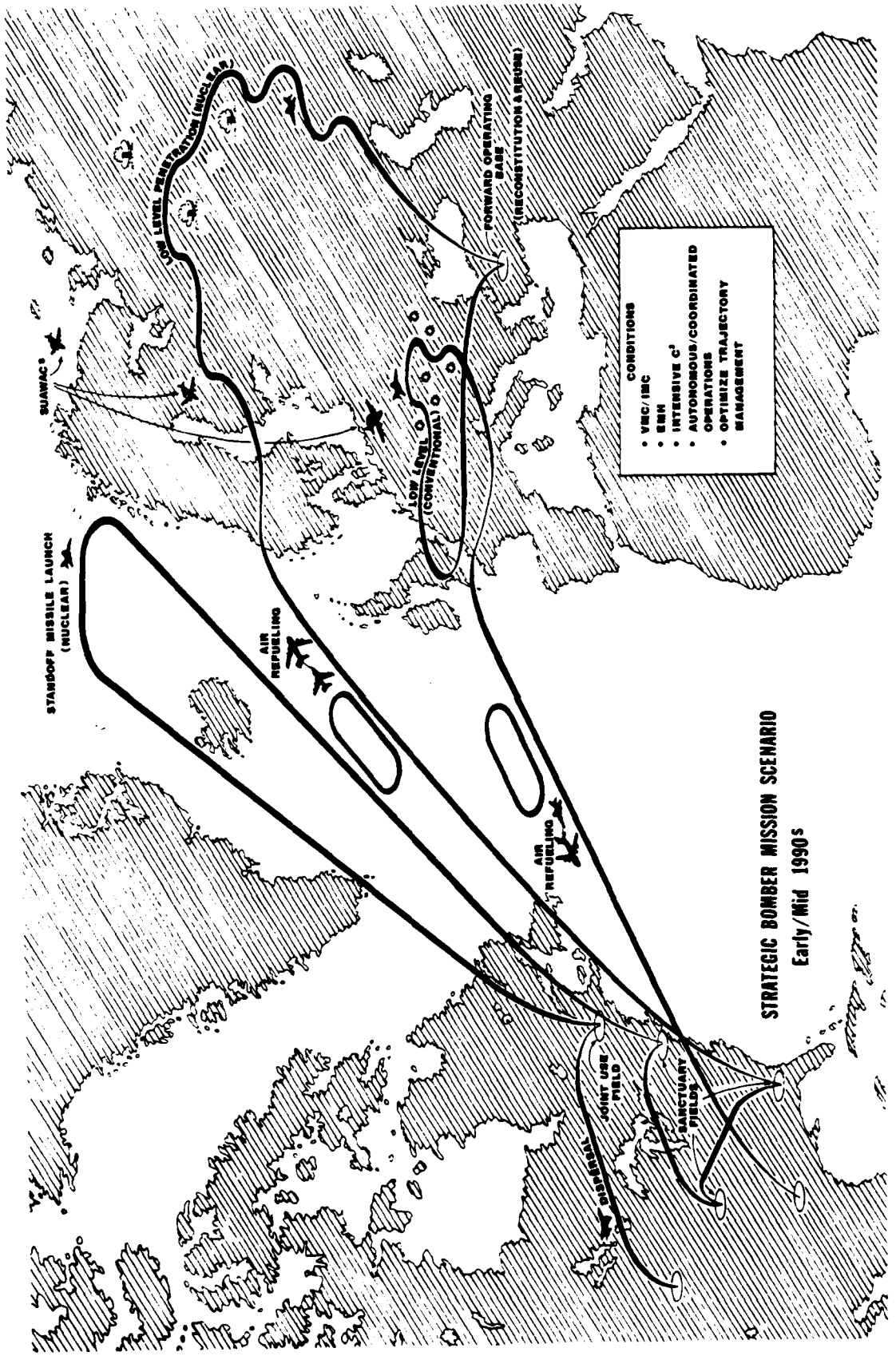
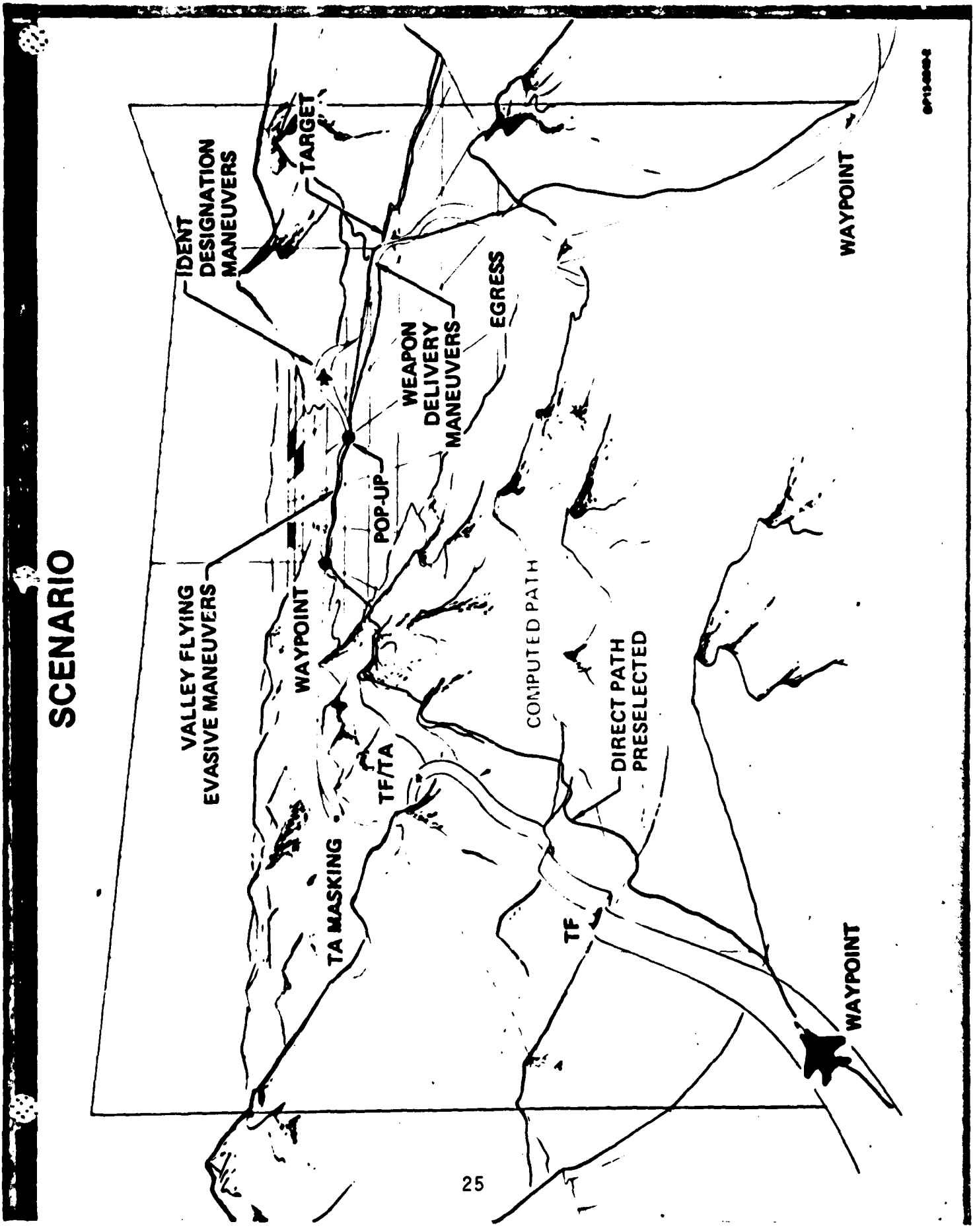


FIGURE 7 - Strategic Bomber Capabilities - Circa 1990

FIGURE 8 - Advanced Concept Fighter Mission



In order to accomplish these missions, tactical and flight related data must be recognized, comprehended, and responded to by the crew. Mission success and safety depend on the timely, and proper, execution of each critical mission phase.

During times of high workload this data can be poorly, or incorrectly, interpreted thus compromising the mission, with the potential for loss of life and property. Methods to improve the tranfer of data to the crew are essential. Once again, graphical displays are being considered as a tool to provide cockpit integration [Mulley, 1980 : 14-15; Ropelewski, 1982 : 39-46].

The Potential for Cockpit Integration

Providing useful, integrated data to a flight crew requires a thorough understanding of the tasks to be accomplished in a given mission scenario. Determining limitations of crews and equipment is a major concern for the Air Force. One way to evaluate the limits within an entire aircraft system is to fly typical mission profiles in simulated combat or emergency conditions. Research studies using simulation have determined the percentage of time the individual crew members spend on specific tasks and pertinent mission phases such as take-off, communications, navigation, radar tracking, etc. These studies identified factors that assist and hinder the crew as they work on tasks. By using

simulation, controlled experimental mission profiles with intense workload situations (such as high communications load, bad weather, and aircraft system failures) have been flown and evaluated to determine the limits of performance of the entire aircraft system [Lizza and others, 1983 : 4-5; Moss and Barbato, 1980 : 2-5; Moss, 1980; Sexton and others, 1976a; Sexton and others, 1976c].

These workload study results indicate that the limiting factor in the cockpit may be the pilot and the inability to deal with overwhelming streams of data. The volume of data which must be recognized and processed in the tactical environment can be too extensive to be properly and consistently evaluated over the course of a demanding mission. Much of the data available requires a significant amount of human processing time to be interpreted. In a worst case scenario with high workload, much data is lost due to a lack of this processing time [Boeing, 1983; Sexton and others, 1976b].

An obvious conclusion is to provide some sort of assistance to the crew by preprocessing at least some of the data. By reasonably integrating the many data elements, the crew will need to process less. Traditional cockpit design does not support this integration well. Often there is little regard for an efficient and effective cockpit interface to the crew even though proper cockpit design is critical if the pilot is to properly use the data available.

Current trends promote the concept of an integrated cockpit. No longer are aircraft subsystems such as the flight control, weapons control, and trajectory control completely segregated. Issues of combining technologies are receiving critical attention. Integrated cockpit design concepts provide essential guidelines for the proper management of the modern cockpit. In an integrated cockpit environment the crew would no longer have to look at several instruments to synthesize a mental picture of what is occurring within and surrounding the aircraft. The cockpit instrumentation would be designed to provide the 'big picture' of the aircraft's situation with a minimum of separate displays.

Electronic graphics display technology offers great promise for integrating cockpit functions. Maturing technology provides an unprecedented potential and flexibility for designing sophisticated aircraft systems based around graphics displays. This inherent flexibility and extensibility of graphics displays provides an almost obvious mechanism for increasing cockpit integration. By capitalizing on the flexibility, display formats can be optimally designed for each phase within a specified mission profile, as well as for many of differing military missions.

Optimization and system integration are not penalty free products, however. The designer of formats faces two rather formidable tasks. The first is the development of 'optimized' formats. Unfortunately there is no quantifiable

measure of optimal. The second task is equally difficult and involves the validation of the format's information content and the proper transferral of that information to the crew member. It is one thing to build an electronic cockpit. It is quite another for that result to work effectively in a mission role. All of these areas require careful study and analysis [AFWAL/Flight Dynamics Laboratory, 1982 : 22-31, 59-69; Curry and Wiener, 1980 : 22-26].

Cockpit Display Evaluation

Government and contractor support organizations are interested in researching and directing the impact that display technology has on future cockpit design. The most notable participants are the major airframe manufacturers (Boeing, Lockheed), key government facilities (NASA Langley, NASA Ames, Air Force Wright Aeronautical Laboratories), and the avionics vendors (Rockwell-Collins, Sperry, Smiths Industries, Ferranti).

The research efforts at the Boeing, Lockheed, and government facilities address three major graphics areas. The areas of interest are the

(1) evaluation of display devices, formats, and physical cockpit arrangements,

(2) evaluation of the limits of automation which can be programmed into cockpit designs and still retain the pilot as an effective system manager,

(3) in-depth study of the advanced display techniques and technologies for both software and hardware portions of the graphics system [AFWAL/Flight

Dynamics Laboratory, 1982 : 59-69; Boeing, 1983; Lizza and others, 1983 : 1-5; Moss, 1984; Sexton, 1983-4; Vokits and Waruszewski, 1980 : 4-8].

The evaluation tool for significant portions of the research is ground-based flight simulation. Real-time, man-in-the-loop simulation is used to investigate the critical areas of interest and to validate the findings. Prior to full simulation, however, extensive design, testing and redesign of display formats must be accomplished. Preliminary evaluations of proposed displays and cockpit designs often take place in graphics support facilities and simulation mock-ups. Results from these experiments feed directly into the full simulation design. This research and development approach dictates that extensive graphics capabilities be available and supported in the various simulation facilities [Holt and others, 1980 : II-1 to II-28, Lizza and others, 1983 : 1-5; Moss, 1984].

Thus graphics development within the R & D community is increasing, with a growing emphasis on real-time graphics for the synthesis and analysis of intelligent in-cockpit displays. These current cockpit integration efforts are certainly a culmination of those efforts which began in the 50's. Today's activities, supported by a much more sophisticated technology base, are more likely destined to succeed. The immaturity of the early technology that plagued the ANIP project will not hamper current work. Vastly improved hardware and software provide a rich set of building blocks to avoid the problems of yesterday.

Graphics Support Environments

Software support for in-cockpit display development has evolved slowly since the early attempts at generating pictures for pilots. The design of support environments for graphics display development is less complete than the hardware support efforts. This reflects the traditional development cycle in which software sophistication typically lags behind hardware advances. Cockpit display support tools are less mature than state of the art support systems for Computer Aided Design/Computer Aided Manufacturing (CAD/CAM) types of applications even though much of the technology is similar. This state of affairs is slowly changing as interest and support for the introduction and use of graphics in the cockpit increases.

There are several efforts whose goal is to provide better design environments with coordination among the various participants. A combined simulation development effort with NASA, Lockheed Georgia, the Triangle Research Institute, and North Carolina State University has encouraged the parallel development of facilities at NASA and Lockheed with the same equipment. This permits a greater exchange of compatible software, thus reducing development time. A coordination activity among Boeing, Lockheed, ... , and the Air Force Wright Aeronautical Laboratories has been working to identify appropriate graphics equipment for use in simulation with the issue of hardware and software

commonality of major concern [Holt and others, 1980 : II-29 to II-42; Lockheed, 1982, 1984; Wall and others, 1980 : 55-59].

The Air Force Aerospace Medical Research Laboratory (AMRL) is researching the area to support cockpit design in general. They currently have a multi-million dollar development program to define a computer-based system which can assist in cockpit design. The Flight Dynamics Laboratory is currently working on a project for Computer Aided Design and Evaluation Techniques (CADET). The facility for display format design would be one of several automated tools within the CADET framework of software support [Bashore, 1983].

The CADET graphics requirements are in turn loosely based on the Panel Layout Automated Interactive Design (PLAID) concept which NASA has used since 1979 at Johnson Space Center to support the space shuttle cockpit panel design. PLAID supports vector only display design by permitting the user to design both primitive and composite objects. These objects are then manipulated by the display processor to effect sizing, rotational, and projection changes in the resulting picture [Rothe Development, 1979 : 1-8].

Another significant effort in the area of cockpit graphics support is an in-house development system, the Symbology Development System (SDS), designed by Kaiser Electronics. This system provides assistance for developing display formats from existing symbology. It is capable of

targeting the final display to the F-18 Heads-Up Display (HUD) hardware. The basic concept of this system is similar to the underlying DESIGNS concept. The actual use of the system has proven to be quite satisfactory and it has grown into a useful tool for that particular application [Kaiser, 1983 : 1-170; Kaiser, 1984a : 1-7; Kaiser, 1984b : 1-8].

There is a rich history surrounding the use of graphical displays for cockpit design. From the earliest programs to the concepts of today, the potential for graphical displays has been recognized. The problems to be addressed now concern how best to realize this potential. Appropriate methodologies and software support systems must be developed to support all phases of the graphics development life cycle.

III. Problem Definition

The life cycle for graphics software development in support of simulation has been discussed earlier in the first chapter. The iterative cycle used while refining graphical designs can be quite time consuming. This is certainly typical of any large scale software development effort. In addition, graphics software development is a very resource intensive activity. Severe demands can be placed on both people and hardware facilities. Actual case histories indicate this resource commitment is both demanding and necessary to accomplish the development. It is intended that software support systems, such as DESIGNS, should address this problem area and provide assistance which would reduce the workload.

The Software Development Cycle

Of critical concern within the life cycle is the level of resource commitment required to support graphics software development and maintenance. Significant graphics software must be designed, coded, and tested to support research simulations. Typically the development of software for display generation requires an extended period of time with the resulting product being generally difficult to maintain and modify.

In addition, a developed software package can be highly machine dependent. As formats are redesigned, much of any previous programming effort must be reaccomplished. New graphics requirements may necessitate the use of a new graphics system. Recoding of entire formats for dissimilar graphics generators can be a monumental task, with the complexity of such an exercise dependent on how different the systems are. The differences in hardware capabilities and software support packages must be accounted for in the recoding.

Efforts are underway to address these needs in the software development cycle. The sheer economics involved in software development is driving both industry and government to develop ways to streamline program costs. The entire area of software design, development, and maintenance is receiving considerable attention.

A Case Study

A contractual effort recently completed by the Boeing Corporation for the Flight Dynamics Laboratory (AFWAL/FIGR) is typical of the resources expended for a graphics intensive project. Boeing was tasked with developing and testing a software package for a set of formats which had been designed in a previous effort.

The format suite contained various tactical situation displays (TSDs), weapon/stores displays, and system/emergency

procedures displays. The proposed suite of formats had been developed earlier by the McDonnell Douglas Corporation. The scope of the effort was to evaluate the formats when they were displayed with differing characteristics. Formats were displayed with color versus black and white presentations, raster versus stroke versus hybrid techniques, and some special graphics presentations such as video disks. These formats were then test flown in simulation using civilian and military pilots. Examples of each type of display developed are shown in Figures 9 through 17 [Jauer and Quinn, 1982 : 22-117].

The facility used by Boeing to produce these displays contained a variety of graphics hardware. The complement of equipment included Sanders, Megatek, Ikonas (now Adage), and Lexidata hardware. Most of this equipment was used in one capacity or another in the completion of the effort.

The manpower for completing the graphics software development involved 3 to 4 senior graphics software people, working in excess of 4 months. Nearly 1 1/2 manyears had to be expended to produce working software for a set of formats that had previously been designed and optimized! Significant project time must be committed to the software development phase independent of the design and modification phases which must accompany every effort.

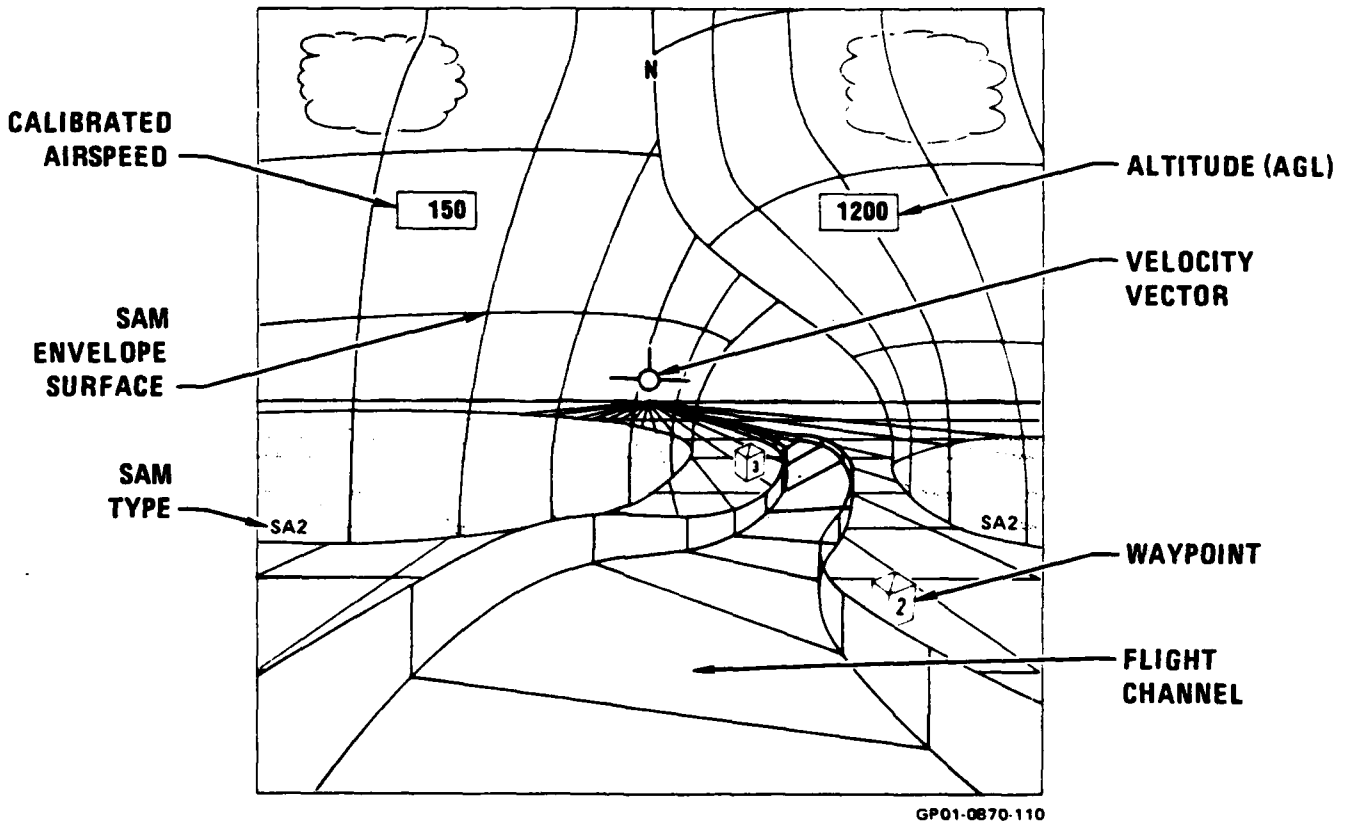


FIGURE 9
TSD in Monochromatic Stroke

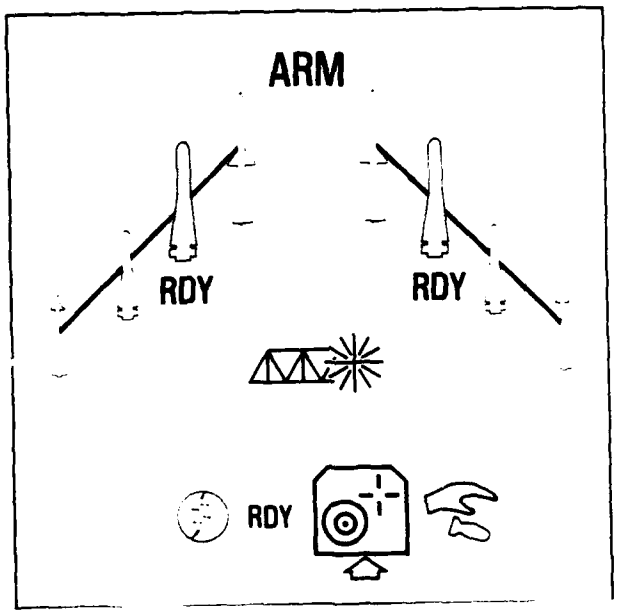


FIGURE 10
Stores Display in Monochromatic Stroke

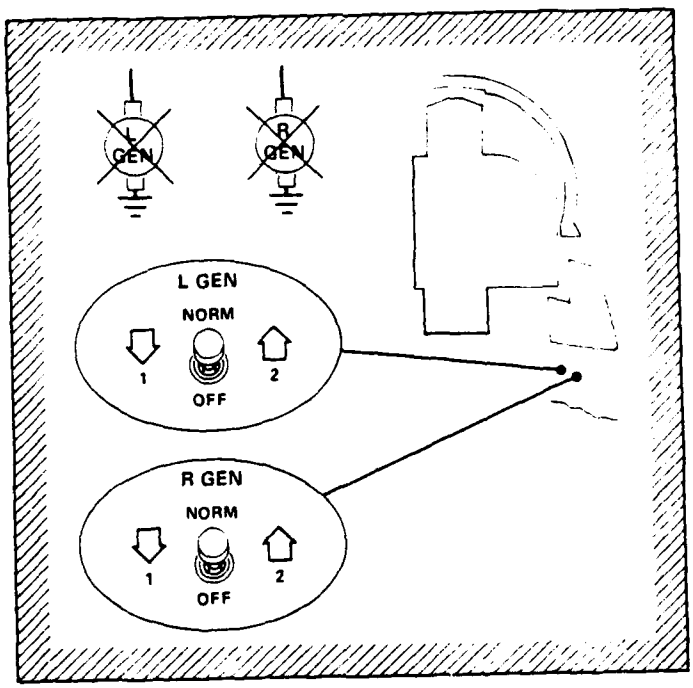


FIGURE 11
Procedural Display in Monochromatic Stroke

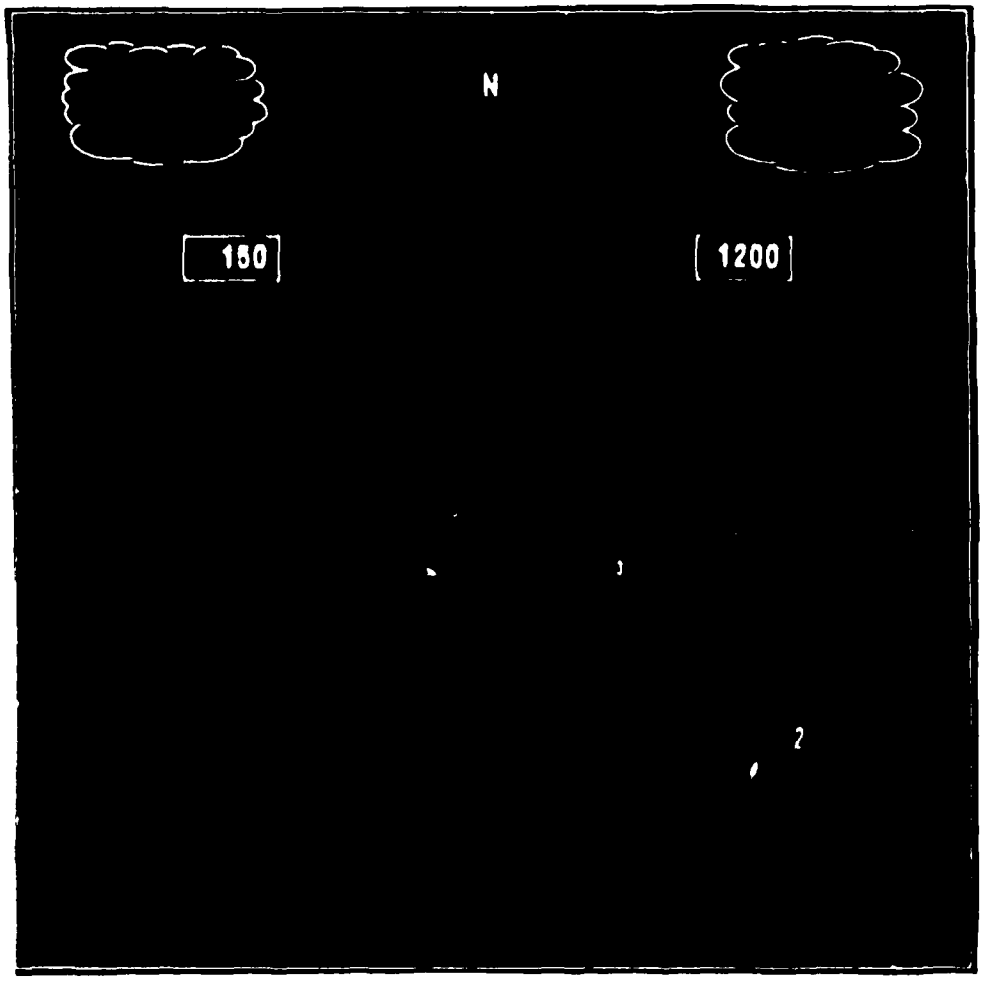


FIGURE 12
TSD in Color Stroke

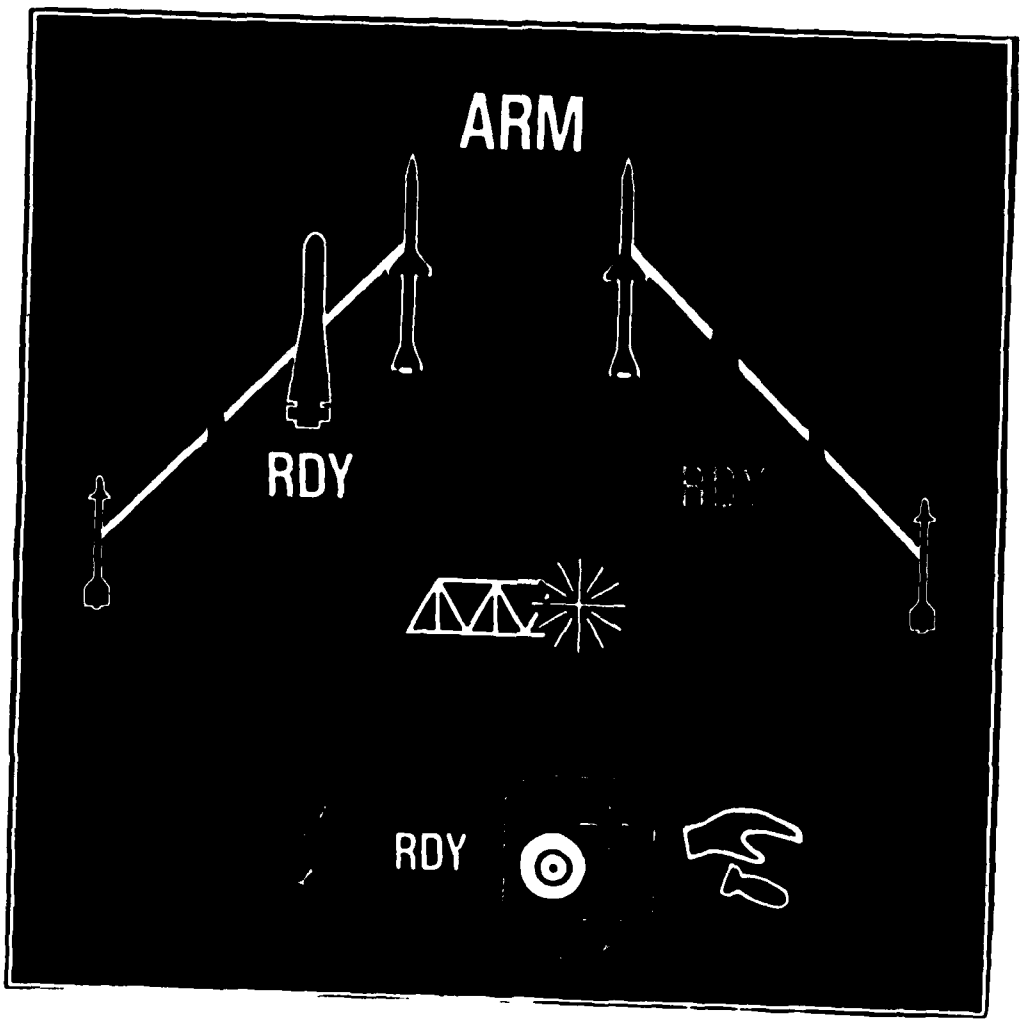


FIGURE 13
Stores Display in Color Stroke

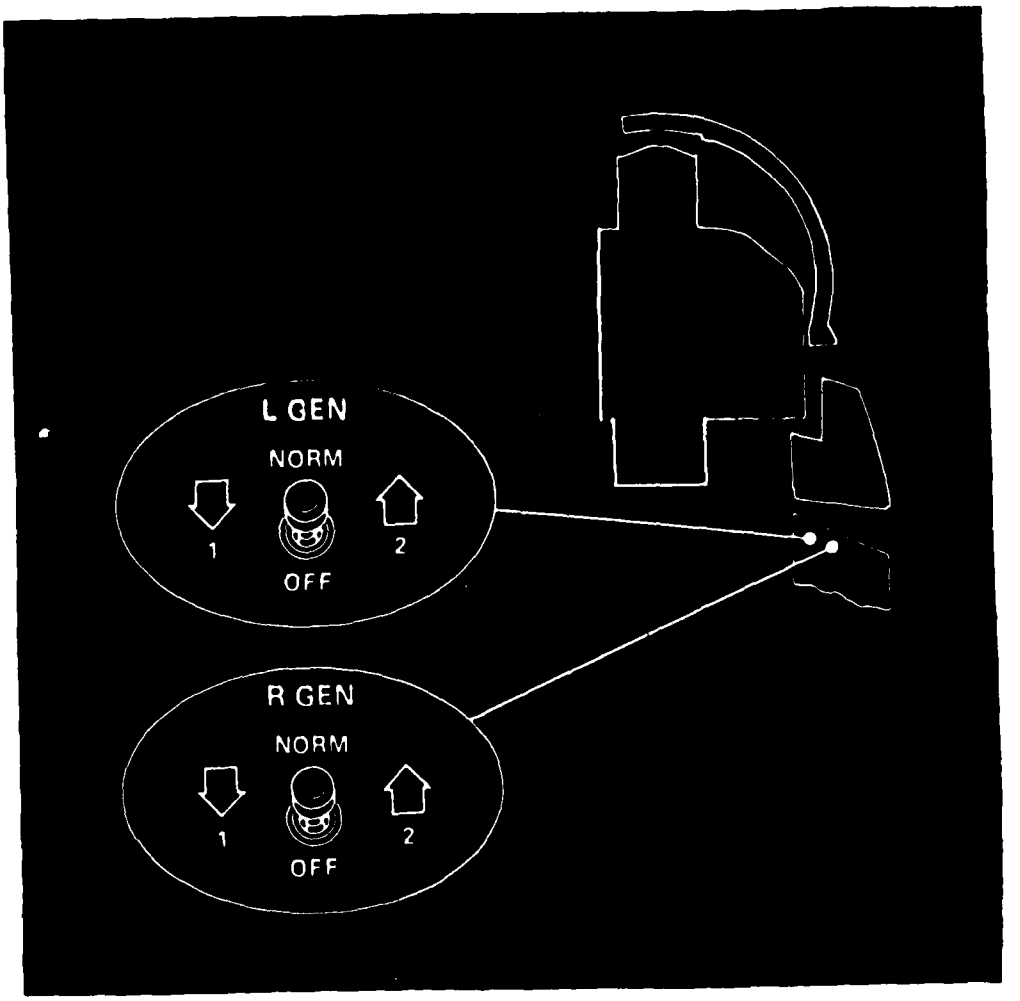


FIGURE 14
Procedural Display in Color Stroke

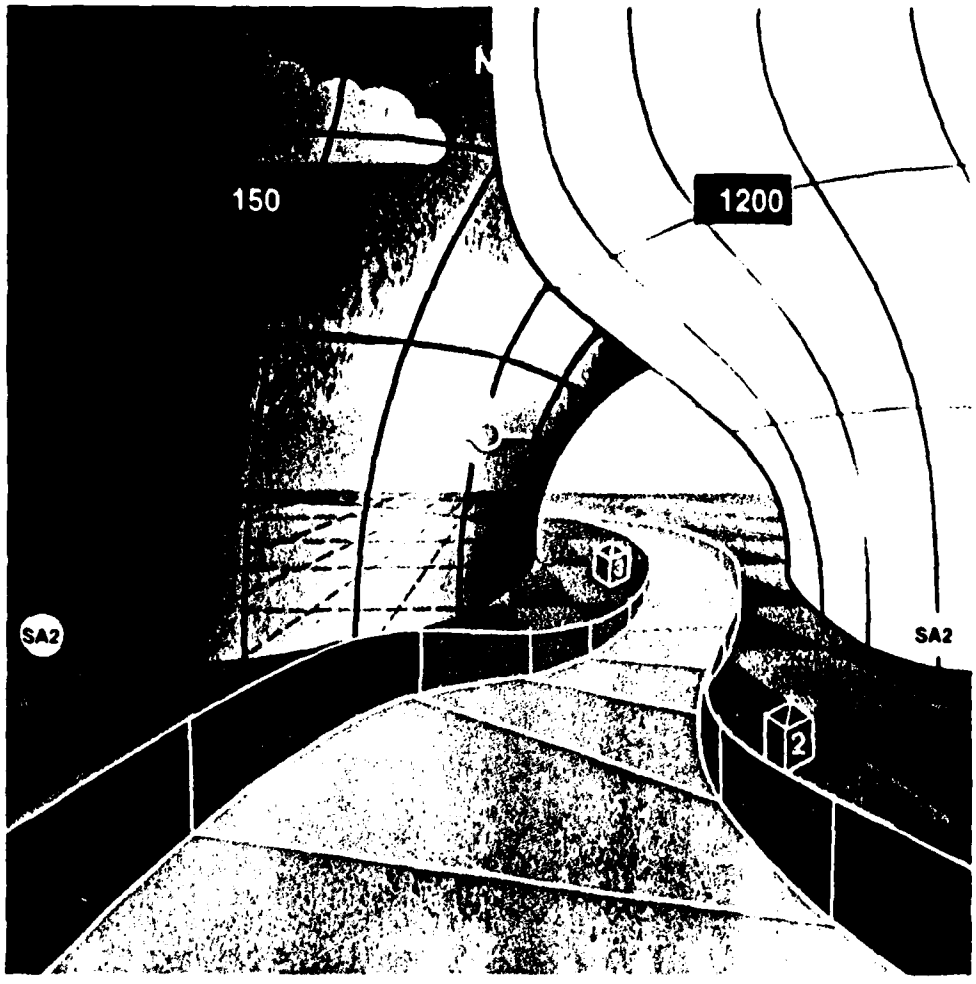


FIGURE 15
TSD in Color Raster

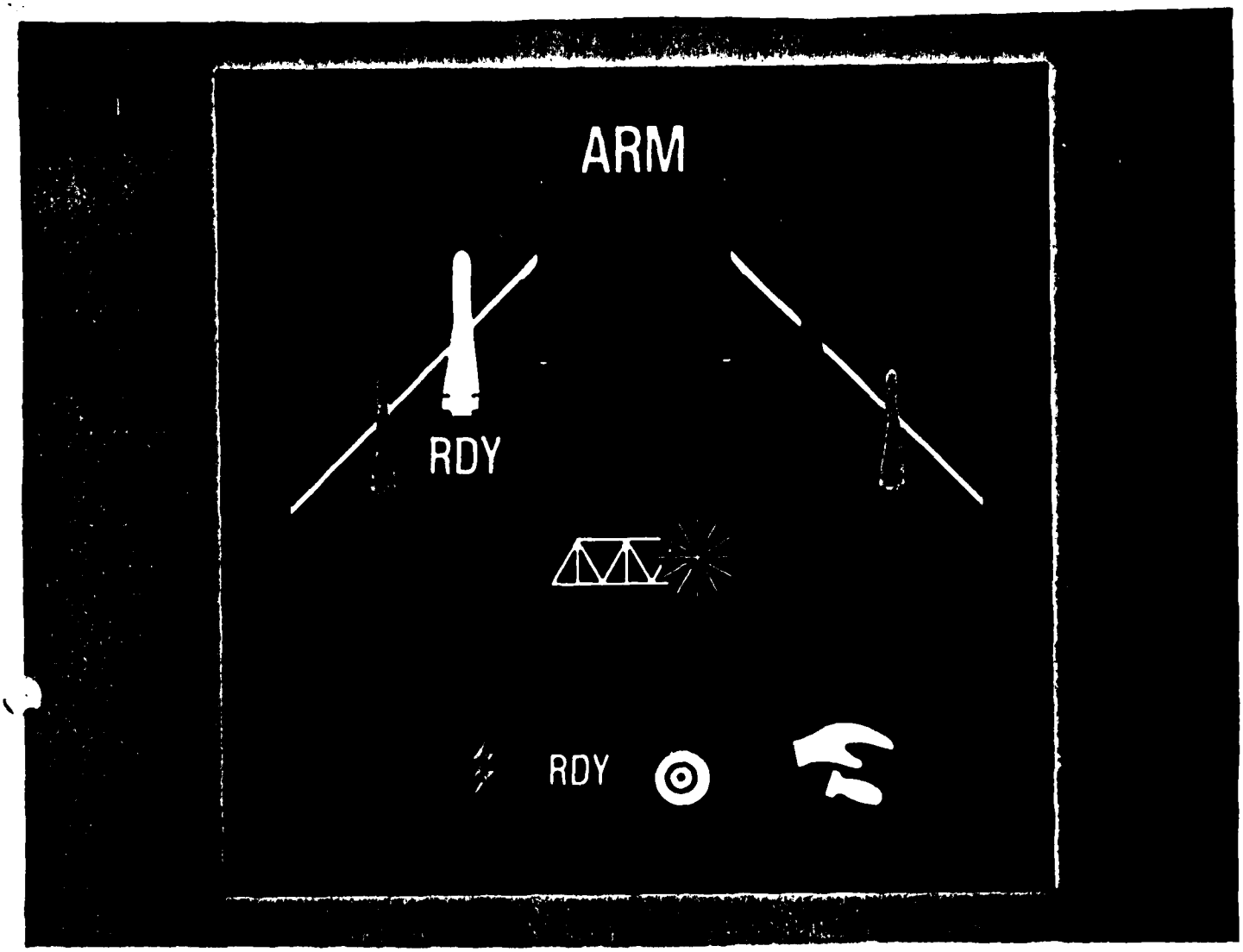


FIGURE 16
Stores Display in Color Raster

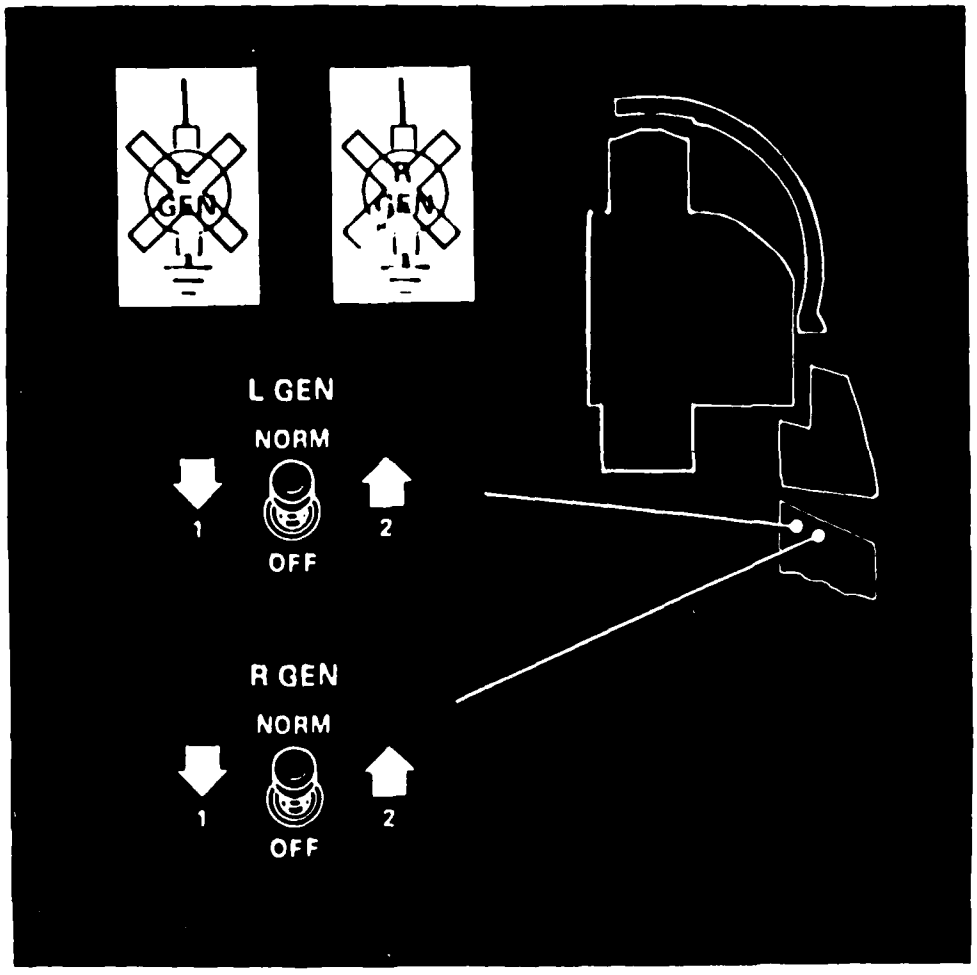


FIGURE 17
Procedural Display in Color Raster

In light of these significant manning requirements it is justifiable to question whether or not the resulting graphics oriented cockpit is desirable. In results from piloted studies, unequivocally, crew responses to graphics displays in simulation have been favorable. Pilots feel that displays provide a natural focal point for integration in the high technology cockpit. Measures of crew performance indicate that there is significant improvement in the overall cockpit management of complex systems and missions. Therefore, it appears that resources expended in this area are well spent [Boeing, 1983; Lockheed, 1984; Moss and Barbato, 1980].

The question then becomes not one of whether or not to develop such design tools, but rather one of identifying critical needs. What kind of tools and capabilities must be provided? Will they adequately and effectively support all phases of graphics work for cockpit displays? To answer this, one must know the actual environment in which the tools are to function. Defining the desired capabilities and operational environment defines the overall problem and identifies the required tools.

The Operational Environment

The Boeing simulation facility typifies the operational environment toward which this thesis effort is targeted. These research organizations have been built up with a variety of graphics equipment to take advantage of the

systems available and to provide a broad spectrum of capabilities. This creates a rich but somewhat imposing environment in which to design and modify graphics software. Compounding the difficulty of equipment variety within a single facility is the ever-present time requirement to develop and test experimental systems as quickly as practical and to support a variety of complex, highly dynamic test programs [AF Flight Dynamics Laboratory, 1981 : 14-29; Becker, 1982-3; Boeing, 1983].

With these research and support oriented facilities, the principal product is system evaluation by means of simulation. The personnel in such a facility are often not the original designers of the display formats. In many cases they have an adequate, but not extensive, understanding of the graphics hardware and software available to them. If the format designer is a part of the development team, this individual is probably even less versed in the technical details of the graphics resources.

The typical facility will have multiple graphics systems of varying types and options. Both raster and calligraphic generators will probably be available. System options are not standardized and often vary with respect to color versus black and white, advanced hardware options such as two and three dimensional transformations, differing display resolutions, and the availability of support software packages.

There is no doubt that the specific characteristics of the environments can be dramatically different from one facility to another. However, there are some key elements each has in common. First, the variety of equipment which may be replaced, modified, or discarded creates a dynamic situation to handle. Second, there will be multiple graphics support packages available as each graphics generator normally has some unique, vendor-supplied package. Third, it is unlikely that personnel will be expert with all the hardware and software available to them. Finally, the time requirements for the typical programs conducted in these facilities demand that complex software development be done quickly. Many of the desired capabilities in a support environment are dictated by these facility considerations [AFWAL/Flight Dynamics Laboratory, 1982 : 12-21, 31-69; Becker, 1982-3; Sexton, 1982-4].

Statement of the Problem

The life cycle issues for software development present a complex set of problems with which the system designer must deal. The issues of concern during this research effort focus on the specific area of support for graphics development in the real-time simulation environment. The many facets of this activity - from the basic issues of format design; the intensive, iterative process of refining display designs; to managing the complex nature of the

facilities available - have been presented to define adequately the problem domain considered. Each of these components contributes to the overall complexity of the problem and certainly molds the definition of potential solutions.

It has been the intent of this research development to provide at least a partial solution which can reduce the amount of time required to develop the actual graphics software required for simulation support. While all the problem areas referenced impact the nature of the solution, the critical question addressed in this effort involved improving the available development tools. The high cost, resource critical, iterative development cycle desperately needs support features to decrease its tremendous impact on overall system costs.

The key problem becomes one of identifying those features, methodologies, and functional capabilities that can positively affect the software development and maintenance cycle. Once identified, actual tools can be developed which improve the users' abilities to accomplish tasks. The goal of the DESIGNS development is to augment the users' capabilities, to provide an alternative, and hopefully improved, environment in which to develop simulation graphics displays.

In order to accomplish this goal, this thesis effort defines the underlying software structure for a set of graphics tools. The resulting software system addresses the

problem defined here by means of an environment/operating system whose function is to support basic display creation and modification. The system centers around an interactive graphics terminal through which the user 'converses' with the DESIGNS software system in the activity of designing or updating a display format. While being tailored to the problematic needs of the Control Sythesis Branch flight simulation facility (AFWAL/FIGD), the definition of DESIGNS does address the general issues required for graphics support.

Constraints of the Problem

A complete solution to the graphics design problem discussed was too extensive to design, implement, and test as a single thesis effort. The objective of this effort has been to address key issues involved in the overall design for the package. In so doing, an appropriate set of tools was identified for development. These tools provided a reasonable subset of the completed system capability, but more importantly provided a stand-alone tool useful for graphics development immediately.

Thus, it was necessary to determine a valid proof of concept approach for the project. This addressed the total system design and implementation issues. In developing the project to prove the feasibility of the concept, a subset of the overall effort was implemented. To this end, constraints

were imposed on the effort in an attempt to limit the project to a manageable size. The resulting effort still yielded a product whose capabilities provided an adequate test of the feasibility and utility of the original design concept.

The primary goal of this thesis effort was to demonstrate that the graphics environment designed for some arbitrary host machine can be used to create and edit display formats, and generate correct source code for a specified target graphics generator. It was necessary to demonstrate this thread through the system to prove the overall utility of the system. Completion of this portion of the design illustrated that it can decrease both the software development time and the re-hosting complexities for graphics software.

One key element in the DESIGNS definition is the symbol librarian. This particular function serves as the link between the editor functions and the system support functions. The complete development of the librarian capability defines the complete thread through the system discussed earlier. Plus it also is a powerful, independent tool. Many formats are based on previously developed symbol sets. A mechanism to manipulate, modify, and selectively use individual symbol definitions in the preparation of different displays provides an excellent basis for the initial DESIGNS effort.

Therefore, this effort concentrated on the design and implementation of the symbol librarian capability. In so doing all the critical areas were addressed, at least from a design point of view, with the exception of image graphics. Complex image generation, such as would be required to depict realistic models for the out the window view, is well beyond the scope of this thesis effort and was not developed further.

To further simplify the system, a single target graphics generator was identified and used for the proof of concept. After demonstrating the concept for one device, additional modules defining other graphics devices can be added to the environment to include any device available as a target. The internal definition of a device driver module will differ for various target systems, since the individual definitions must be tailored to the hardware and software capabilities specific to the given device.

The contention in the development and testing phase of DESIGNS is that the proof of concept can be demonstrated with a single target device. At a later time device driver interfaces can be installed for other systems. To demonstrate that the single target is not being handled as a special case, stubs for other potential targets will be included. However, for the initial thesis effort, these stubs had minimal functionality other than to demonstrate that the design accounts for multiple target devices.

This effort did not address the issues involved in designing the dynamic test capability into the environment. A dynamic testing function built into the DESIGNS environment would provide a method to check out symbol dynamics prior to actually interfacing with a simulation. However, the critical need within the simulation facilities is to create the original format designs, many times evaluating a series of them in static, mock-up conditions. While dynamic testing is a valuable extension to the project, it is not necessary to prove concept utility. This enhancement would, at some future date, improve the overall usefulness and power of the system.

In part, the driver for selecting the set of restrictions for the effort came from a consideration of the potential end-users of the thesis product. The thesis development effort, as defined with its limited scope, can serve as a solid starting point for many of the graphics efforts involved in aircraft simulation. By consciously designing for the anticipated system enhancements and extensions, the software package can evolve into the complex environment described earlier.

Selection of this baseline set of capabilities was based on the apparent short-term needs of a specific simulation facility. The AFWAL/FIGD and FIGR facilities have an immediate desire to use such a support tool. Their initial needs are in the area of heads-up and head-down displays which use alphanumeric and simple symbology in the formats.

Continued discussion and involvement with these organizations helped to narrow the focus of the project. With this guidance, a more appropriate system was designed to meet the critical needs of today. By considering future requirements in the definition phase, the DESIGNS system can continue to expand to meet future application needs.

Thus the scope of this thesis has been delimited to demonstrate a complete path from the format design and editing in the host system, to the display generation in the target graphics system. The core of requirements includes the design and implementation of a symbol librarian for storing and manipulating symbol definitions, with design considerations for the editing of these symbols/icons, the definition of the interfaces to the host's operating system and the target device, and the assignment of dynamic attributes to elements within the display format.

IV. DESIGNS Requirements

It becomes apparent that there is a need for graphics design tools which provide a working environment powerful enough to support the designer's requirements for display format design. Such a support environment should provide the display designer with a 'user friendly' interface to the available graphics design tools. It should provide a straight-forward mechanism for both the initial design and the subsequent software maintenance and modification. In addition, it is desirable for this system to have the capability to target the final display to any of several target graphics generators. These capabilities greatly decrease the time it takes to design or modify cockpit display formats. Such a design environment does not require that the designer of the formats be knowledgable about either the graphics software or hardware. Thus the human factors designer can be designing the format and the software to display it at the same time.

Synopsis of Requirements

The highest level requirement levied on the DESIGNS definition is that it carefully consider the environment for which it is intended. It is NOT strictly a graphics endeavor. Major portions of the system requirements specifically address specifications for the user interface.

Remembering that the key goal of the effort is to simplify the software development and decrease development time, the nature of the user interface is critical for success.

Certainly of equivalent importance is the set of tools that can be provided within the DESIGNS toolbox. These tools provide features which can assist in graphics design, modification, and maintenance. The third major portion of the system is the link to available hardware devices. Not only must the target graphics systems be considered, but also the DESIGNS host, and the user workstation.

Graphical requirements are certainly an important element in the overall effort. Throughout the various subsystems, graphical issues are addressed. The graphical capabilities that are required are implemented with reference to existing standards and graphics techniques.

It is necessary that the system software that results from this work be a consistent, maintainable package. To help accomplish this, it has been required that structured design methodologies and the accepted practices of top-down, structured programming be used during the development of DESIGNS.

Referring again to Figure 5 in the first chapter, it is apparent that these three areas - user interface, toolbox, and interpreter - define the infrastructure for the DESIGNS environment. The graphics requirements are interwoven within each subsystem and certainly direct design decisions.

The complete, integrated environment must provide extensive supporting tools to use the results of the graphical data manipulation in an operational simulation. This family of tools makes it possible to transition the developed graphics technology into the exact configuration required by a simulation - whether that be as an integral part of the simulation displays, an off-line analysis tool, or a stand-alone design tool. The DESIGNS definition postulates tools to integrate and support the various available graphics processors, the user interface, the dynamic test facilities, and the necessary host processor capabilities. The following sections discuss the functional requirements of DESIGNS within the context of these subsystems.

The DESIGNS User Interface

The user interface for DESIGNS is the focal point for the effort. The underlying package of tools must be designed for use by the non-expert in graphics as well as the expert programmer. Thus, the interface between the user and the package must be carefully planned to permit users with a variety of backgrounds and levels of expertise to use the system with equal effectiveness. The system should provide a 'pleasant' environment for the user, one that is relatively straightforward to use. At the same time, the DESIGNS tools must be extensive enough to provide an ample user interface

to all of the system's capabilities.

The typical user of the proposed DESIGNS package will be an engineer or human factors psychologist. There will be no dedicated operator for the system, thus all users are in a sense 'casual'. There is no required minimal level of competancy in computer science. The user should only need to know how to initially start the system. Thus it is important to provide a system which is easy to understand. Any user should be able to engage in a dialogue with the system to create, save, or modify display formats. No formal understanding of any graphics drivers or hardware is necessary. All the interfaces to actual physical devices are invisible to the user, thus imposing no in-depth knowledge requirements.

With this definition for the system user, it becomes apparent that the techniques used to define the 'conversation' between user and computer must support ease of operation. Interactions with the system must be clearly defined from within the dialogue itself, so that the user is dealing with an autonomous system.

The importance of the definition of the user/machine dialogue in the DESIGNS development can not be overstated. The dialogue design and implementation bias each user either favorably or not toward the final system. The dialogue provides the mechanism to access the tools defined for DESIGNS. Therefore, the level to which the dialogue can be user and application tailored will determine the success of

the overall package.

Emphasis is placed on designing an autonomous system, relatively simple for even the uninitiated or infrequent user to operate. The system should provide a mechanism for expansion and adaptability to support the research environment for which it is intended. The needs and capabilities of the DESIGNS facility and users provided the impetus for the design process. The resulting dialogue reflects those needs and thus, in all probability, will be inappropriate for other classes of users or operating environments.

It is required that a menu driven approach for the dialogue be used based on the definition of the DESIGNS user. For the environment in which DESIGNS is to be used, the major concern is to make it available and usable to a variety of infrequent users. The ease of use criterion is very important.

Thus the relative ease with which a menu driven system can be used offsets the potential shortcomings. The menu driven dialogue implemented for DESIGNS does consider ways to avoid some of the pitfalls of the classic menu driven systems by using some command language features. This somewhat 'hybrid' approach enhances reliability and user acceptance. Appendix B, which discusses the human factors techniques involved in user/machine interactions, presents the pros and cons of the various dialogue methods available.

The DESIGNS Toolbox

The system toolbox must provide an adequate set of graphics tools to accomplish the tasks of development and modification of simulation related displays. The toolbox must also support the overall life cycle of the graphics development by means of facilities to interact with available host tools, dynamic testing functions, and documentation support.

A critical function of this set of tools is that of editing display formats. The user must be able to create and modify graphical files in a manner analogous to the familiar editing of text files for other areas of software development. The graphics editor supports the initial design and subsequent changing of alphanumeric and symbolic displays.

A fully implemented editor supports the more complex area of image design. Images are graphical representations of actual scenes, such as a synthetic terrain display. Development and modification of such displays requires more extensive tools in the area of modeling, shading, hidden line removal, textural features, and other advanced graphics techniques. Table I defines in more detail the desirable features an editor must have to meet these functional capabilities [Pavlidis, 1982 : 17-19].

In addition to editing graphics symbology, the DESIGNS environment must provide a mechanism to describe the

TABLE I

Functional Capabilities of the DESIGNS Graphics Editor

T A S K	C A P A B I L I T I E S
Text Editor	Text entry Text deletion Text replacement Text repositioning
Symbol Editor (predefined icons)	Symbol entry or selection Symbol deletion Symbol placement Symbol repositioning
Symbol Editor (undefined icons)	Symbol drawing/design Symbol modification Symbol entry into library for system expansion
Image Editor (pictorials)	Extend drawing techniques Support arbitrary format design Painting capability

characteristics of the symbology. Complete symbology definition includes both the physical appearance - such as shape or size of the icon, and other describing characteristics - such as location, color, intensity, etc. An object is not completely defined until the characteristics are specified. An object's characteristics can be modified over a period of time. Therefore, it is also necessary to define the possible dynamic transformations a given symbol may undergo. Table II details the functions which must be

supported for the complete definition of a symbol's attributes.

TABLE II

Assignment of Static and Dynamic Attributes

T A S K	C A P A B I L I T I E S
Define fixed attributes	Define position (x, y, z) Define object size Specify color/intensity Define orientation
Define transformations	Dynamic translation/rotation in 2 or 3 dimensions Alphanumeric updates Variable color/intensity Variable perspective
Identify the dynamic elements	Tag dynamic alphanumeric fields Identify dynamic symbols Identify those transformations permitted on an element

Once defined, the various building blocks of a display format need to be maintained for future use and modification. As previously discussed, much of the effort in graphics development lies in the use and modifications of existing symbology. The DESIGNS system proposes a symbol librarian function which manages the database of graphics symbols. The librarian must provide the interface into the database for the user to manipulate the graphical definitions contained therein. Table III defines the tasks for which the symbol

librarian is responsible. The tasks of the symbol librarian parallel those of an operating system file manager. Details concerning the types of display formats to be supported by the librarian will be discussed under the simulation topics chapter.

TABLE III

Functional Capabilities of the DESIGNS Symbol Librarian

T A S K	C A P A B I L I T E S
Addition/deletion of the definition for symbols	Access database definition for the library to add/delete any specified symbol
Updating existing symbol definitions Store multiple versions if desired	Interface with the graphics editor (predefined icons) Provide audit trail of changes made to designed symbols
Automatic linking of the source code modules to create formats defined from previously tested symbology	Retrieve specified symbols Link the various modules for a complete file for the static display Create the dynamic update file based on selected attributes

In addition to these tools supporting graphical functions, there are several general support facilities that a sophisticated environment should offer. These extend beyond the graphical issues of format design and consider areas which impact the overall life cycle considerations for the software being generated. Table IV lists the type of

support features required in the complete system.

TABLE IV

Integration and Support Areas

SUPPORT AREAS	FUNCTIONS
Host computer operating system interface	Access to the system tools, functions, and storage media System error checking
Dynamic Test Facility	Dynamic format demonstration Format evaluation and debugging Offload eventual target device
Documentation Facility	Document display requirements Document new symbology Audit trail of display mods

An important part of the toolbox support is the provision for an interface to the host computer's operating system. This interface provides the necessary system functions to support the graphics design process, but does not require the user to understand the detailed working of the operating system. The DESIGNS system should be able to exploit the useful features of the operating system that are available.

Another support function required within the DESIGNS toolbox is the dynamic testing facility. A sophisticated system should provide some method for dynamically testing the formats which have been developed. This permits the user to

demonstrate the dynamic behavior throughout the range of operation so that any anomalous behavior can be detected. Undesirable effects can be corrected in the design system prior to generating code for the target system. Such a dynamic demonstration provides the final test assuring a complete and correct implementation of the original design specifications.

One final support tool required in a complete system is some form of on-line documentation. It is important that the user be able to describe and track the work that has been accomplished with DESIGNS. Having the documentation available within the graphics development environment improves the definition of the symbols and formats defined. It also provides a more cohesive development tool since all phases of the development work can be accomplished using a single support package - namely DESIGNS.

The DESIGNS Interpreter

The use of the term interpreter is somewhat misleading in the DESIGNS conceptual model. This block within the model represents not only the mechanism which interprets DESIGNS directives to produce target source, but also the definitions of the target hardware and software, along with all the interfaces required for DESIGNS. Much of the DESIGNS interpreter software must be developed to provide adequate definition of the available hardware so that the full

potential of DESIGNS can be realized. Thus it is imperative that the complete system requirements and final design consider the desired synergy of the software and hardware components. Table V lists the functional capabilities of the interpreter subsystem.

TABLE V

Functional Capabilities of the DESIGNS Interpreter

T A S K	C A P A B I L I T I E S
Define target hardware and software support	Description of the available hardware options; definition of software call structures
Generate source code for static display	Describe 'rules' governing the software structure required on a given graphics target
Support source code for run time modifications	Provide template for dynamic updates Provide means to modify the effect of symbol dynamics by altering drive equations
Define user workstation	Describe the workstation capabilities Define standard interface to the capabilities

The important requirement for all of the interpreter capabilities is that all interfaces between the graphics design environment and ANY physical device can, and must be, defined in terms of the data exchanged between them. In

DESIGNS, this interface specifies the format for the communication between the modules. The final step of the interpreter subsystem is then a device driver of some sort which is specific to the device and accomplishes the interpretive phase of the system. Thus any device driver, while its internal structure is highly device dependent, simply conforms to the pre-established communication format in order to exchange display data. Thus if the interface definition is complete, any target definition can be added to the system. Proving the concept requires thorough testing of this interface definition to check its adequacy.

This concept of device independence - whether it is for the target systems or the user workstation, is important for several reasons. It permits extension of the DESIGNS capabilities to new target devices. This extension can be provided in uniform manner using device dependent routines only at the lowest software level. It also provides for better portability by isolating the device dependent modules.

Since the benefits are several, the DESIGNS interpreter subsystem must be defined with the issues of extensibility and device independence in mind. The graphics tools and the environment which they provide must be versatile enough to permit targeting to any of several available graphics systems. At the same time, it must be extensible to permit additions to or modifications of the existing interface package in the event that the graphics capabilities change. This extensibility must include changes to both the graphics

hardware and their software support packages. To accomplish this adaptability, the DESIGNS environment must define the target machines. Such a definition should be in terms of the graphics software packages which are available on the target device, reflecting those hardware functions which may be accessed on any given device. Such a system definition supports the generation of source code appropriate for a specified target environment. Thus source programs for any number of dissimilar target graphics systems can be generated from the generic format descriptions defined within the design package.

The DESIGNS environment should embed the device dependent structures in internal modules, the structure and existence of which the user need not know. From the user viewpoint there is only a single structure to manipulate and that is the DESIGNS database which is accessed through the user interface. External target machines can be selected by the user, but require no further user definition.

It is imperative for the successful implementation of the DESIGNS concept that a standard definition for an interface to multiple targets be developed. Without some standardization in the target definition, both the hardware and support software differences for varying targets become major problem areas for DESIGNS. The aquisition of new target devices or the upgrading of old ones would require tremendous modification to the DESIGNS system if large portions of the software package were constrained to being

target specific.

The issues of concern in this area of interface definition cut across the boundaries of both software and hardware and demand a thorough appreciation of each. In order to effect the design of such an interface, individual target machines must be known as part of the DESIGNS environment by means of common system descriptors. This system definition includes information specifying the hardware and software parameters of the specific system. Each target then has its own definition module internal to the DESIGNS package. The important implication here is the complete development of adequate abstract data types for the target machine definitions.

The actual target definition must include data which completely defines both the hardware available and the software which supports it. Much of the hardware definition can be used to assist the user in selecting an appropriate final display source if such a decision has not already been made. The hardware definition must include such data as the number of output channels available on the system, whether it is raster or stroke, color or monochromatic, how many colors or intensity levels are supported, whether it has hardware or software supported transformations, has multiple processors, or software overlay capability. Using this information, the user can intelligently select the target machine best suited for the application at hand.

The definition of the target support software should be completely invisible to the user. This information hiding is consistent with the policy of not requiring the casual user to be more knowledgable about the technical aspects of the system than necessary. This internal definition defines the available software library callable subroutines, procedures, and functions which support the target options.

The issue of commonality among the target descriptions also impacts the future extensibility of the DESIGNS environment. Common definitions imply that multiple targets can be available during any given format designing session, thus providing the user with a variety of potential target devices. Targets can be added or removed from the DESIGNS database by creating or deleting the definition structures. In this fashion the DESIGNS software can be upgraded to reflect the current facility capabilities without massive changes to the underlying package. The DESIGNS software remains unmodified, only the target definition package requires changing.

Once a standard structure for the target device hardware and software definitions is defined, all systems can and must be specified in the same fashion, requiring no specialized software tailored to the individual target system. The user is at liberty to use raster or calligraphic target machines interchangeably, needing only to be aware of the aesthetic differences in the resulting displays.

Just as new targets can be added by creating an initial hardware/software specification, so also can these specifications be modified. Upgrades or changes to any existing graphics processor require that the database defining the machine be modified to reflect the new functions. This update assures continued compatibility with the target configuration. The capability to interactively update these definitions has previously been mentioned under the support areas of DESIGNS.

The DESIGNS Facility

It is required that the design and software developed from this effort be compatible with the thesis sponsor's facility. The Flight Control Development Laboratory in the Air Force Wright Aeronautical Laboratories (AFWAL/FIGD) is typical of the military research facilities and is to be the end user of this research product. Historically this facility has been tasked to support HUD and in-cockpit displays for several fighter aircraft such as the F-15, F-16, and the AFTI-F16.

Experimental displays for non-conventional aircraft such as the Forward Swept Wing airplane have been supported. Design work for display algorithms to support Terrain Following/Terrain Avoidance (TF/TA) missions is a major activity. In addition, support for a variety of navigational heads-down displays with moving maps and flight planning data

have been supported. The symbology for all of the displays has been designed and displayed on a monochromatic, calligraphic display. To date this is the only graphics capability in-house.

Currently an advanced color, raster system is being integrated into the existing simulation computer facility. With the completion of that installation, more complex display generation can be supported. With the advanced hardware in place, heavier emphasis will be placed on pictorial formats. Facility planning calls for the supported growth of the graphics power by further additions of equipment in the following years [AFWAL/Flight Dynamics Laboratory, 1982 : 35-53]. This serves to increase the need for a standardized design environment which can support the different pieces of graphics equipment.

Based on the current in-house graphics capability and the projected efforts to be supported, the critical graphics task is the development of calligraphic heads-up displays for Air Force fighter aircraft and fairly simple heads-down displays. No sophisticated pictorials or color symbology can be supported at this time. However, these have been considered as extensions to the environment during the design phase.

For the short term, the facility requires design support to develop displays for the type of cockpit configuration depicted in Figure 18 [General Dynamics, 1983 : 8-2].

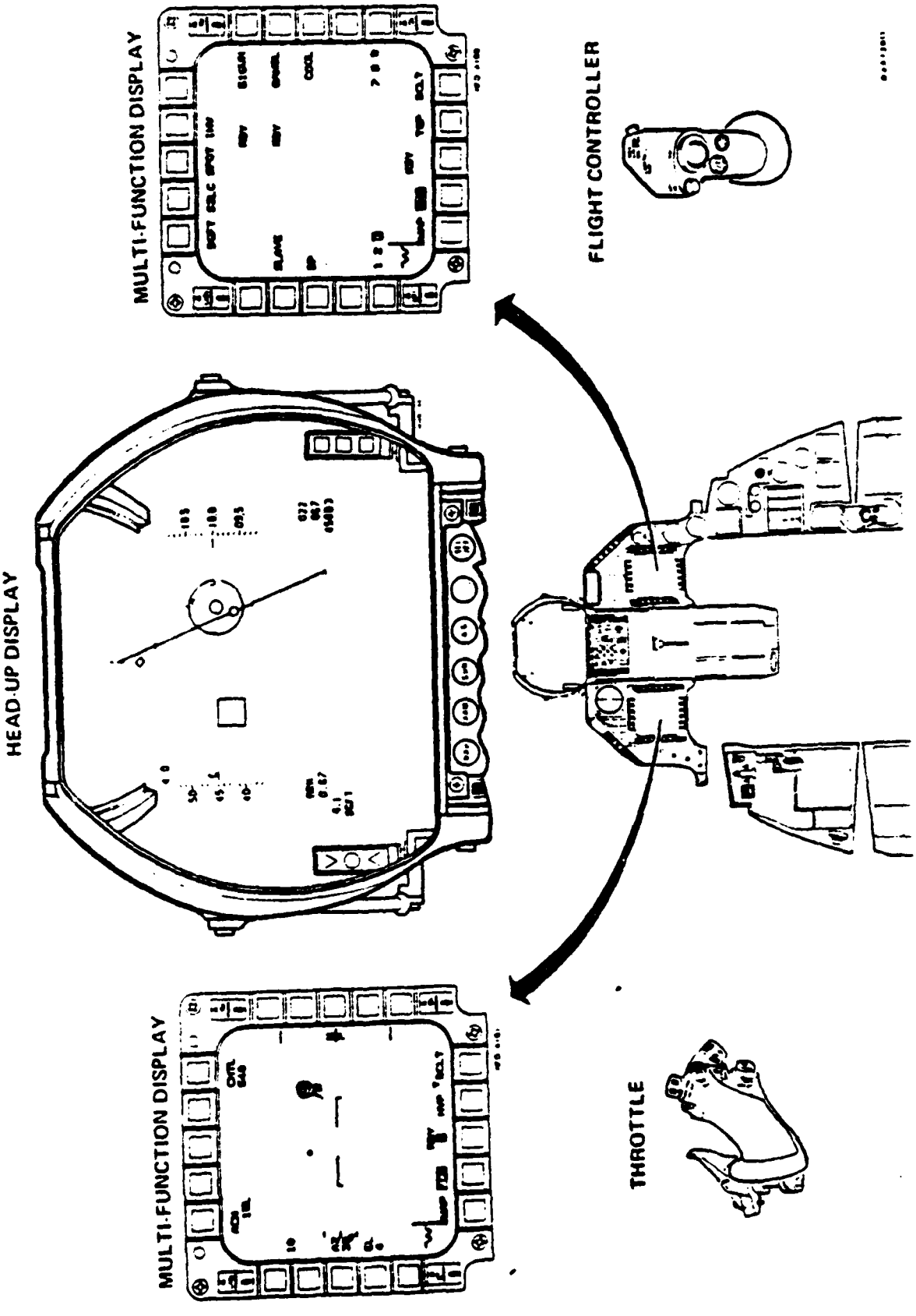


FIGURE 18
Typical Cockpit Display Configuration

The design environment must provide the interface to a single target graphics device as it is the only one immediately available for simulation support. The long term needs dictate that multiple devices of differing capability and type must be defined and integrated as targets. Furthermore, the complexity of display formats will increase, requiring the expanded editor capabilities discussed in the last section. It would be beneficial to implement the dynamic testing facility as support for any long range activities.

The discussion of the long term growth in the simulation facility indicates the importance of the extensibility of the effort. If a truly useful tool is to evolve, it must be developed with change and growth anticipated. Identifying future needs, and defining them to whatever degree possible during the initial project development, strengthens the possibility of expansion.

Summary of Functional Requirements

In summary there are six major functional areas in which a complete, rigorously developed environment must provide functional capabilities. These provide all the services required by the user to fully develop, debug, and maintain graphics software. They must include the capability to

(1) perform editing of static graphics formats,

(2) assign static and dynamic attributes to specified display elements,

(3) provide access to existing libraries of symbology,

(4) provide the interfaces required for the user workstation,

(5) support the interface to the target systems, via the target definition packages, and

(6) provide access to host computer utilities, support dynamic testing and documentation.

To support these features, the DESIGNS host system, consisting of a user workstation and host computer, must have the following capabilities. The workstation must have

(1) a full alphanumeric keyboard, preferably with programmable, function keys available,

(2) a joystick or mouse with some picking capability,

(3) the potential to add a datatablet in the future to digitize pictorial formats,

(4) a single display screen with a minimum of 2047 x 2047 addressability,

(5) a support software package that can be accessed from a Higher Order Language (HOL) that supports the generation of graphics primitives such as lines, characters, and conics, with some form of display list segmentation permitted,

(6) the ability to display multiple intensity levels as a minimum, with color preferred for the long term,

Preferably, the workstation would be a raster system. For the longer term concerns, a raster workstation permits the design of both calligraphic and raster formats better than a stroke only system would support raster targets. However, the basic DESIGNS concept does not require that the workstation be raster capable.

The host processor for the DESIGNS system likewise can be described. Based on the functional requirements

discussed, the host computer must have

(1) an HOL available, preferably Ada although the structure provided by Pascal would support the development well; FORTRAN is acceptable, but does not have the overall power and structure of the other languages,

(2) appropriate compilers for the HOL,

(3) an editor, file manager, and link/load capability as a minimum; the Ada Programming Support Environment (APSE) would provide a much more powerful and complete development environment,

(4) access to hard disk mass storage, the size of which is only limited by the number of different formats one wishes to design (reasonably, 10-20 MByte maximum should be required),

(5) access to magnetic tape storage to provide a backup capability for the system,

(6) access to an adequate amount of physical memory (estimated at 128K words), supported by the capability to program with overlay structures, or task activation, so that unused portions of the environment need not always be resident in memory, and

(7) access to physical interfaces with all the target devices (preferably this interface is a direct one).

There is nothing specific to the DESIGNS requirements that mandate that the host be a mainframe system. Mini- and micro- systems designed as workstations are certainly adequate if they provide the access to the required devices. There is also no requirement that the system be multi-user, although the design does not preclude this.

In order to develop software to satisfy the functional requirements of DESIGNS, certain software engineering standards had to be met. The techniques of structured programming were used throughout the development effort. The

use of pseudocode, or a Program Design Language (PDL), in the actual development of code was used. This supports good design practice and serves as adequate in-line documentation for the system software produced.

Additional tools from the theory of structured analysis and structured design have been employed during the analysis and design of the effort. The SADT methodology has been used in concert with pseudocode in the design and development process of the system [Rutledge, 1982; Softech, 1978: 1-1 thru 1-2].

A derivative of the SADT methodology developed by Softech in the early 1970's was used to accomplish the functional analysis and basic system design. The SADT method was selected for use even though it is more complex than other analysis tools available such as structure charts. The SADT was selected since it, and its derivative, have been used as a tool in the laboratories in the past. This derivative was developed a few years ago as the design tool of choice within the sponsoring organization [Softech, 1979: 4-1 thru 4-23].

These design descriptions serve also as tehnical documentation for the DESIGNS effort. They should define the system well enough to support future modications and extensions.

The testing of DESIGNS has also been based on proven techniques. Due to the overall size of the project, exhaustive testing was impossible, so boundary checks,

special case testing, and functional module testing have been important testing tools [Rutledge, 1981; Weinberg, 1979: 171-176]. The tools developed have been tested as modules to assure compliance with design goals, and then integrated with other tools to produce a larger, integrated system.

The following chapter presents the design and implementation of the DESIGNS environment. Special emphasis is placed on those data structures and tools that have been implemented to date. An overview of the complete system is also included.

V. The DESIGNS Implementation

The preceeding chapters and the appendices have provided a great deal of information which is supportive of the DESIGNS development process. The characteristics of the DESIGNS environment have grown out of the ideas and techniques presented in these chapters. While the supporting technical information has been somewhat lengthy, it was necessary to provide the understanding of the groundwork on which DESIGNS has been built.

The DESIGNS concept is that of a robust, ever-growing environment whose sole task and reason for being is to support the graphics development work that is accomplished in the Flight Dynamics Laboratory's simulation facility. Its initial identity is one of a set of loosely related tools which have helped to automate the task of graphics software development. This initial set of tools is primarily aimed at supporting the development of HUDs from libraries of existing symbol sets.

It is a developer's aid, a support facility whose ultimate realization will be a completely integrated environment providing the necessary features for advanced graphics generation across a broad range of equipment and capabilities. The laborious attention to detail in the technical discussions has been critical to identifying and defining the capabilities that will be needed to actually realize the complete DESIGNS potential.

The entire DESIGNS implementation is specific to the operational environment and users for which it was created. The repeated attention to device independence and system expansion draws a focus to the critical fact that the DESIGNS support system is functioning within a highly dynamic environment. Its functional requirements have been specified, and its growth capabilities postulated, to provide reasonable means to grow and adapt within the dynamics of the facility.

This chapter discusses the important features of the actual facility in which the DESIGNS system functions. It relates the necessary details of the development practices which were in use prior to the advent of some of the DESIGNS capabilities. Most importantly it defines how the DESIGNS system has been developed to meet the requirements which were levied on the system. The chapter focuses on those portions of the DESIGNS environment which have been implemented, tested, and are currently in use within the simulation facility.

Facility Description

The facility in which DESIGNS resides is the simulation facility within AFWAL/FIGD. This facility has been discussed in general terms in Chapter 4. As was described there, this organization is actively involved in real-time, man-in-the-loop simulation.

The heart of this facility is a pair of Systems Electronic Laboratories (SEL) 32/77 mainframe computers tied together via a shared memory interface with a SEL 32/2750 mainframe. The SEL 2750 is the host processor for a proposed suite of graphics processors. It currently supports two Megatek 7000 series calligraphics systems whose primary function is special purpose heads-down displays. A third calligraphic system, a Vector General System 3 (VG), is also available within the facility. This unit, hosted by an Electronic Associates, Incorporated (EAI) PACER 700 minicomputer, is the facility workhorse for the HUD development. The PACER 700 is interfaced to one of the SEL 77's through a dedicated digital to digital interface (DDI).

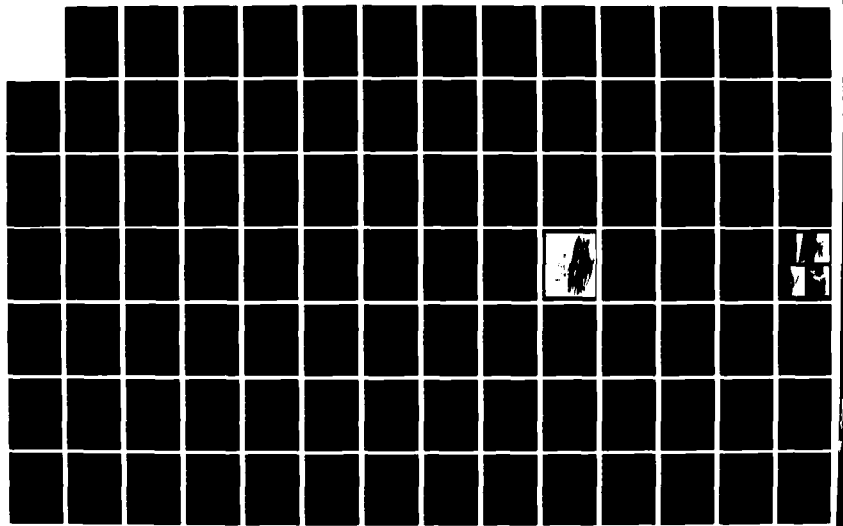
The SEL machines are supported by the MPX operating system. It is a multi-user, multi-tasking, real-time system which supports FORTRAN 77+ as its only high order development language. The PACER computer runs a proprietary hard disk operating system supporting a single user, writing FORTRAN IV based software.

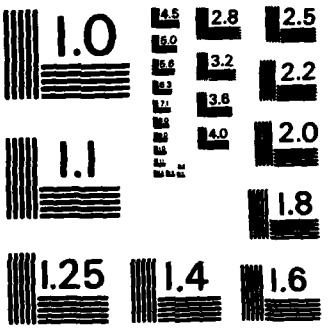
The Software Development Cycle

The normal procedure for developing graphics software within the facility depends on whether the Megatek or the VG hardware is to be used. If the Megatek is to be used, the software is created, modified, and maintained directly on the host processor, the SEL 2750. All the editing tools,

AD-A159 218 THE INTERACTIVE GENERATION OF ALPHANUMERICS AND SYMBOLOGY WITH DESIGNS ON THE FUTURE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 2/3

UNCLASSIFIED K A ADAMS JUN 85 AFIT/GCS/MA/85J-2 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS - 1963 - A

compiler and cataloging modules, and storage facilities are available directly on the SEL. Typically each format, or graphics application, is developed independently. There is minimal coordination among the efforts and very little, if any, standardization.

The procedure for developing on the VG is more complicated. The host machine for the VG is strictly dedicated to running the compiled software and is not a development system. A second PACER 700 machine must be used to compile and link software modules in preparation for both testing and simulation use. The PACER systems, primarily due to their age, do not support a reasonable editor or any sophisticated development tools. Therefore, source software for the VG system is created in the SEL 77 machine, using its more powerful editor. This software is then transferred to the development PACER through the DDI for compilation and linking. The executable module is then tranferred by moving the removable disk to the PACER hosting the VG.

This process is extremely time consuming, especially when there are errors in the software. To correct the errors, the disk must be moved from the VG host back to the development system, corrections made in the SEL editor file, the corrected source is re-transferred to the PACER, re-compiled and linked, and the newly created executable module on the disk taken from the development PACER to the VG host PACER. The user literally runs in circles as the software is being initially developed.

As with the Megatek software, traditionally each HUD development was an autonomous effort. Even though basic symbology was duplicated from one simulation to the next, the software was pretty much written from scratch. This situation was an outgrowth of the way the VG software packages were written to handle display list pointers for dynamic updating. With the traditional way of handling the pointers, a great deal of manual labor was required to make changes to the display list itself. These changes were not only tedious, but extremely prone to error. Coupled with the circuitous route necessary to create new software, development time for VG software could be quite prolonged. Development time in excess of several weeks was not unusual.

In keeping with the initially stated goal of this thesis effort, the implementation of the DESIGNS capabilities has focused on improving the productivity within the existing organization. DESIGNS is the key element to achieving that improvement. As an outgrowth of the DESIGNS considerations, a high level executive, which provides loose integration and a focal point for all end-users of the graphics tools, has been developed. It provides the framework for not only DESIGNS, but all current and future tools in the facility.

DESIGNS itself is a much more potent tool. While it must function within the hardware and software systems that exist in the facility, it offers the potential for much more in the future. The emphasis has been to improve the development and user environment, to minimize some of the

tedium involved with the PACER development, and to provide a single focal point for the facility graphics development. To this end the overall DESIGNS system has been postulated, and a functional subset of tools, with particular emphasis on the critical PACER/VG problem, has been implemented.

The DESIGNS System Specification

It can't be emphasized enough that the DESIGNS system is quite specific for the development of simulation software. The formats developed using any subset of the DESIGNS system are not merely developed and modified within the DESIGNS environment. The display list information generated MUST be tranferred to the selected target. This resultant display definition MUST then be accessible to a running simulation. The display formats are dynamically updated based on the responses of the aircraft system in reaction to pilot inputs. It is this issue that differentiates the DESIGNS system from other graphics development environments. DESIGNS is not an end in itself, it is merely the means to achieve an end. The 'end' in this case is a fully functional display, running on the desired graphics device, which can be dynamically modified from a simulation, running in a different computer system.

The design specification of the DESIGNS system will be discussed referring once again to the familiar Figure 5 from Chapter 1. This figure provides the point of reference to

focus the discussion. The description of the design progresses from the user interaction with the system, through the editing tools and the interpreter, to the actual target devices. Thus the user's perception of the system activities during a development session is preserved.

This is certainly not to imply that DESIGNS was developed in this order. In actuality, several of the lower level tools were designed, tested, and used as stand-alone entities, long before the actual user interface was designed and implemented.

The DESIGNS User Interface

The selection of the hardware portion of the DESIGNS user interface, the host computer and the user workstation, was made from the equipment available at the FIGD facility. The two systems considered were the SEL 32/2750 with the Megatek units and the PACER/VG system. Based on the requirements of the preceeding chapter, the SEL/Megatek combination was selected. The PACER/VG system was discounted for several reasons including,

(1) the older FORTRAN IV language support, and, in general, poor software development tools,

(2) the lack of adequate memory, the PACER supports only 32K words, and minimal disk storage capability,

(3) the VG has NO available input devices, it is strictly output only,

(4) the age of the system (over 15 years) does not make it an attractive choice for a potentially

long-lived development effort, and finally though not as important,

(5) the system is heavily committed to actual simulation support, thus has only a small amount of time available for development.

The SEL/Megatek system meets, or exceeds, all requirements. Its only major drawback is that it has no direct interface to the VG, which is an important target device. It can access it indirectly through the SEL 77 system, which is acceptable, though not optimum.

The Megatek , in its current configuration, does not support color, but has 16 levels of intensity. It has a fairly robust software support package providing for multiple line types, various character sizes, hardware transformations capability, and segmentation of the display list into a maximum of 32 separate pictures. The package, though dated by today's standards, was one of the earliest to be based on the principles of the CORE Standard. In addition to the software support, the Megatek has joystick, datatablet, function keys, and special interrupt (picking) devices as well as a full ASCII keyboard. Thus it provides a fully acceptable system to serve as the user workstation.

The SEL 32/2750 computer system serves as a very capable host. It supports all the required development tools. Although the language support currently on the system does not include Ada or Pascal, the FORTRAN 77+ that is available is more powerful than any FORTRAN implementation that this author has used. The SEL language implementation provides

excellent structuring features including,

(1) Do While and Do Until structures,

(2) Select Case capability, and

(3) a Datapool structure, which is a vastly improved common structure that is not dependent upon the declared order of the variables, as is the case in the more familiar FORTRAN common. The Datapool usage is dependent upon the spelling of the variable name.

In addition, the SEL machine has access to over 300 MBytes of hard disk storage, two 9-track tape drives, and 256K, 32-bit words of memory. The I/O capabilities of the system more than meet the requirements for DESIGNS.

The DESIGNS system embraces the philosophy that a helpful, computer initiated conversation is the most supportive for its users. In keeping with that philosophy, the DESIGNS dialogue is completely menu-driven. Both alphanumeric and graphical menus have been used. The use of graphic symbology, where appropriate, has been used to provide stronger support to the user. For most applications, intelligent users prefer a graphics representation of the information. With a graphical format, data is presented at about the speed at which the user can absorb it [James, 1981: 389].

Within the DESIGNS environment, each menu level available is presented in the same manner. This consistency is maintained to aid the user. At all times the user has available certain key information which defines the state of the system. This information includes,

(1) the actual menu from which selections are made,

(2) access to HELP information relevant to that menu,

(3) an area in which the current format can be created or modified,

(4) a status area, indicating what menu is currently active,

(5) a standard option to exit the current menu, and

(6) a scratchpad area in which the user may type commands and answers to prompts, or the DESIGNS system may write prompts and HELP messages.

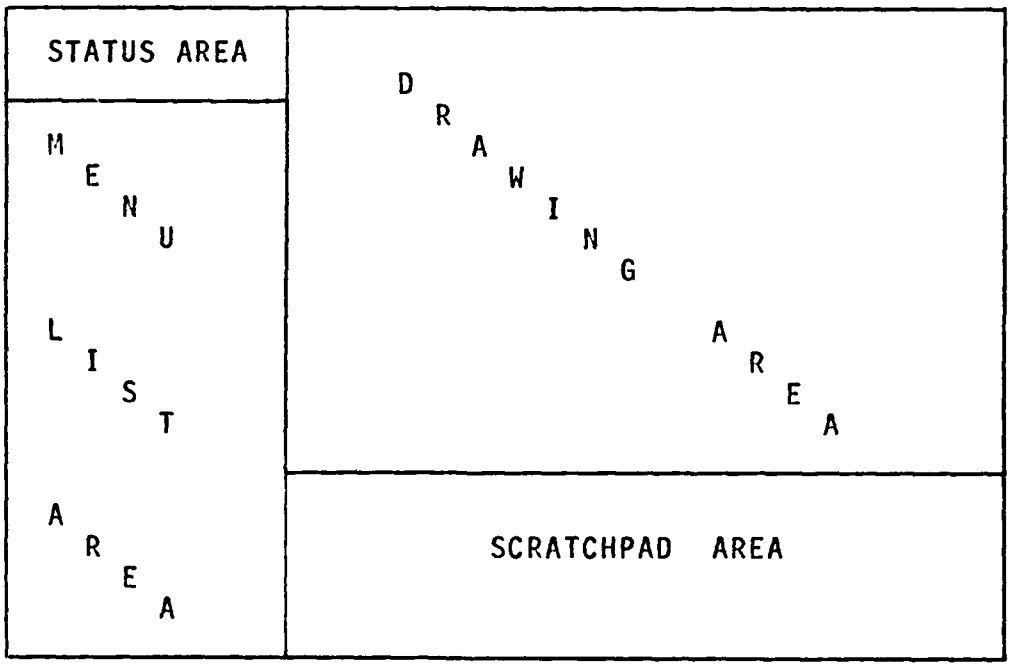


FIGURE 19
The Generic DESIGNS Menu Format

Figure 19 illustrates the generic form of the menus presented to the DESIGNS user showing the locations of the items listed. The individual menu items may consist of alpha strings or actual graphical symbols. Symbols have been

employed at the menu levels in which a symbolic representation is more descriptive than a word or phrase. Figure 20 illustrates one such symbolic menu.

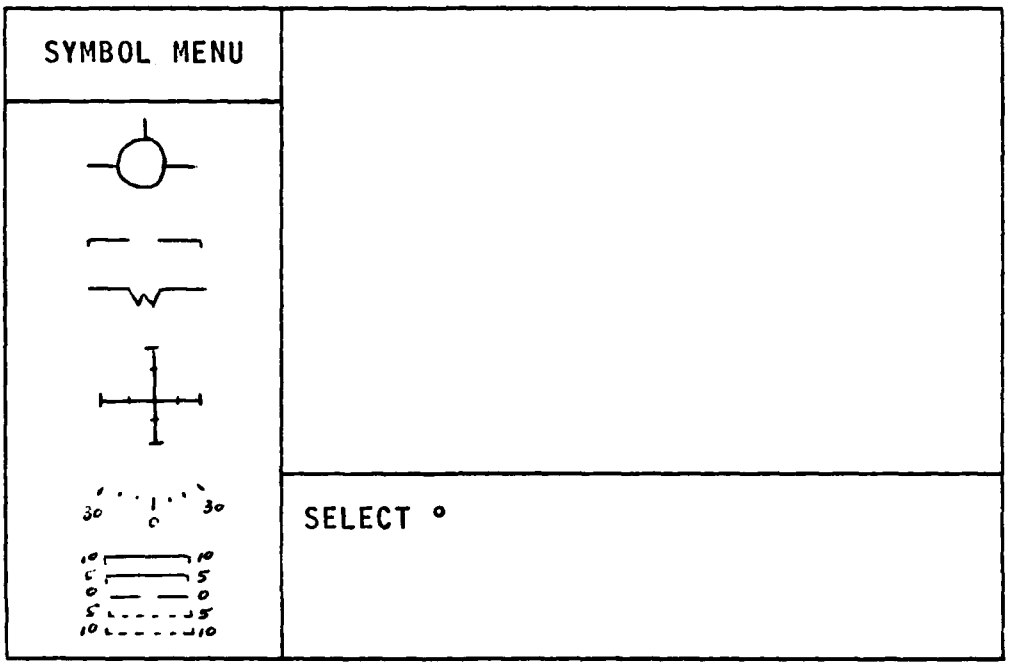


FIGURE 20
A Typical DESIGNS Symbolic Menu

Menus using word descriptors are short, usually two or three words. They describe the activity which will be initiated upon selection. Each entry consists of a verb describing the action, and the object on which the activity will be performed. Figure 21 shows the Main Menu options within the system, and is typical of the style used in descriptive menus.

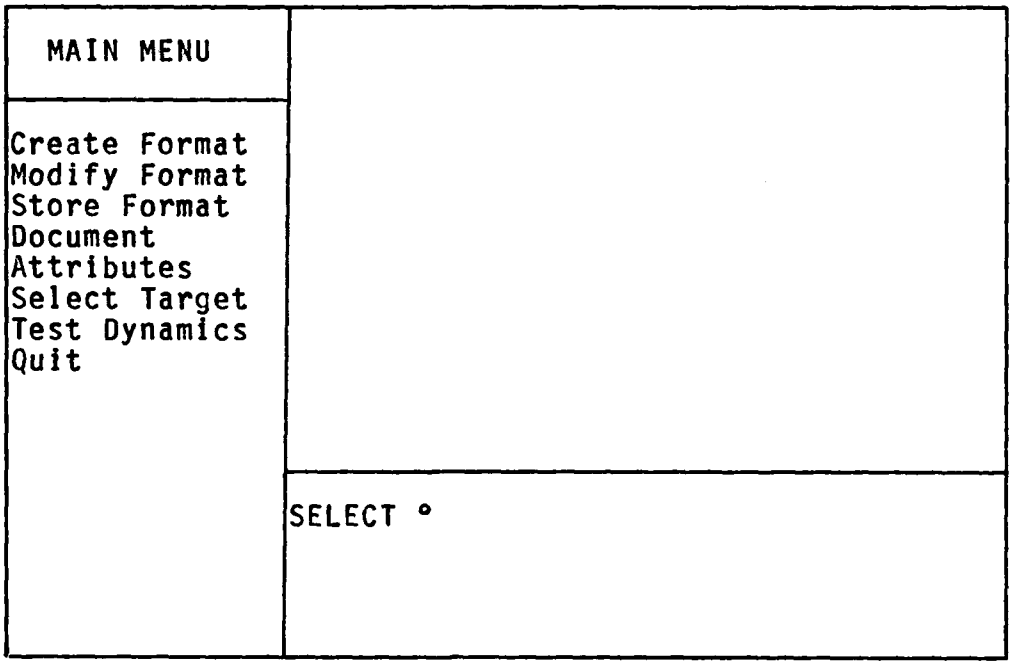


FIGURE 21
The DESIGNS Main Menu

The selection process for all menu items is identical. The Megatek joystick is used to position a cursor on the desired item. Once properly positioned, the interrupt ('picking') button on the joystick is pressed to signal the actual selection. Assuming the cursor is positioned within a valid area, the selection is made, and the activity initiated.

A short circuit to this menu selection activity is provided. The user may also select menu items by typing an appropriate command into the scratchpad area. The form of the command to select menu items is an alpha string. The system recognizes the shortest string which differentiates

between the possible selections on a given menu. Longer strings, specifying the complete command are also syntactically correct.

Referring to the Main Menu, the following short example explains the use of the short circuit. To select the 'Create Format' option, the shortest, syntactically correct, command is the letter 'C' by itself. This uniquely identifies the desired command. If 'CREATE' is typed in its entirety, that also is correct. Compare this to selecting the 'Store Format' option. To differentiate this item from the 'Select Target' option, at least the 'ST' must be typed. An 'S' by itself does not uniquely identify any available menu item. The key to using the short circuit successfully is recognizing that any substring of the option, that uniquely identifies that option, will properly select it. In addition, the full string will also select the option.

The short circuit is also provided for the symbolic menus for those users who prefer to make all their entries from the keyboard and not use the joystick at all. Here the name of the symbol, or the identifiable substring, may be typed into the scratchpad. This short circuit is provided to improve system response time. This is especially important to those users who have become familiar with the system and find the joystick selection method too cumbersome. The user's guide in Appendix F defines the acceptable commands within the system and provides sample session activities to illustrate how DESIGNS can be used.

The DESIGNS Toolbox

Within the structure of the DESIGNS toolbox, there can be a great many functions. It was beyond the scope of this thesis to provide the design for all such possibilities. Several key functions were designed and developed in detail. Others have been considered in such a fashion to facilitate their addition at a later time.

The graphics editor, coupled with the symbol library functions are key to the initial development of DESIGNS. The design for the graphics editor focused on the first two task areas defined in Table I, editing text and predefined symbols. For each of these editing types, the ability to enter information at a designated place on the screen, delete that data, and move it within the drawing space is supported.

Once an editing function has been selected, a positioning cursor appears within the drawing area. This cursor is moved vertically, horizontally, and diagonally within the space by means of the joystick. The user moves the cursor to the desired location of either text or symbol. In the case of text entry, the user would start to type at this point. In the case of a symbol entry, the symbol type is first selected via the symbol menu selection precess, then 'deposited' at the selected location by again pressing the joystick button.

In the deletion mode, the alpha string or symbol is selected. Once selected it is removed from the display being

built. Selection varies depending on whether it is text or symbol deletion. For text, the joystick is moved to the beginning of the text to be removed and the button depressed. This identifies one corner of the area to be blanked. Then the cursor is moved to the opposite/diagonal corner and again the button depressed. The user is asked to confirm the deletion. Upon positive confirmation, the text bounded by the window created is blanked out.

For a symbol deletion, the symbol is picked by means of the menu selection process. When the delete symbol option is selected, a secondary menu appears which lists only those symbols currently a part of the format. The user then selects from that list the item to be deleted. As with the short cut provided to intially add symbols to the picture, the user can also type the delete command followed by the desired symbol name and that symbol will be deleted.

Repositioning of text and symbology follows the same precedural form as deletion. The text area or symbol is specified as in the deletion process. Once selected, it is physically moved by moving the cursor to the desired new position and pressing the joystick button.

The same process of selection is used to select alpha fields or symbols when specifying the attributes of each. Once selected, the user is presented with a series of questions from DESIGNS. These questions lead the user through the definition process for the selected object. For the specification of x,y coordinates for the objects, the

DESIGNS system provides two methods. For HUD specifications the user may define positions in milliradians of displacement from the HUD boresight. This is the typical way HUD data specifications are made. For heads-down displays and HUDs in which positional data is not defined in milliradians, the user may specify relative positions in normalized coordinates. Once specified, the display being defined is updated to reflect the positional data just defined.

As the display is being created, a data structure internal to DESIGNS is created which defines the format. This description of the format can be stored as it is for later use, or further processed by the interpreter section of the system. The actual data structure is quite simple. The basic structure contains item descriptors with the associated x,y location. The descriptor in the case of a symbol is the name of the symbol, and in the case of an alpha field is the string itself. Additional data defining the attributes of the items is stored also. The attributes which are currently handled include,

(1) scale facter,

(2) intensity or color of item,

(3) rotation angles, and

(4) an indicator specifying that the item can be dynamically modified.

If the item has dynamic characteristics, the following functions are supported,

(1) alteration of x,y, position and/or angles of rotation,

(2) turning item on/off,

(3) modifying the intensity/color,

(4) blinking the item, and

(5) changing the value in an alpha type field.

To support dynamic updates, a secondary data structure consisting of a set of logical flags is maintained on each item which can be modified.

The functions within the symbol librarian and part of the interaction with the host processor tools are not directly accessible to the user, but are used internal to DESIGNS. The librarian functions are accessed at the time that the target device is specified. Based on the definition for the format that has just been defined and the definition of the specific target, the librarian converts the format definition into the series of commands that will create the display on the target device. The host processor interface is important during this phase as it provides the mechanism for storage and retrieval of display lists.

The portion of the symbol librarian designed for this effort deals with the definition for the VG target system. The specific context is that of HUD development. Within the library there is a specification for each defined symbol on the VG system. For each additional target there must be a similar specification for the device. The details of the VG specification are given in the next section.

Once a HUD format has been designed, the user normally would specify the target graphics system. The generic display list definition defined previously becomes the input to the interpretive portion of DESIGNS. The namelist of symbols is traversed and the corresponding definition specific to the VG is retrieved from the library. For this initial specification of DESIGNS, the symbol library for the PACER/VG system has been defined as a set of FORTRAN IV source modules - one for each defined symbol on the PACER system. The details of the data structures are presented in the next section.

As the namelist is traversed, the proper module for static generation of that symbol is retrieved from the library. All these modules are combined to produce a single, compilable program for the PACER/VG target. In addition the dynamic routines necessary for each symbol are also generated. Thus the result of this activity is a complete, correct program to be compiled on the PACER for the VG target. The only manual programming activity required is to transfer the software to the PACER, compile it, and create the dynamic module which communicates with the simulation.

During this activity there is a great deal of interaction with some tools already developed on the SEL system. A previous design effort within the facility produced a package which permits a very compact storage of source programs on the SEL disk. This tool, Source Library (SOURCLIB), has been used as the storage mechanism for the

DESIGNS symbol libraries. Some tuning of the package to make it better suited for the DESIGNS system were effected. These changes deleted the interactive portion of the basic SOURCLIB package and made it directly accessible to DESIGNS [Luhrs, 1983]. Many of the I/O functions of the SEL support system are also used to store and retrieve information.

Two other tools within the DESIGNS toolbox have been discussed in previous sections. While neither the dynamic test capability nor the documentation facility were designed to any level of detail, some discussion of them is important. The documentation facility is to be built around some of the existing capabilities in DESIGNS. The capability to enter textual data can be extended to support documentation. A detailed plan of how such documentation should be used will dictate the type of query capability, and database structure that is necessary to adequately support it.

The dynamic testing issue was not addressed in any detail during this effort. The data structures specifying those attributes which can be modified already exist within the system. The tools must be added which can access those structures and modify the appearance of the format being manipulated.

Critical Tools Implemented

As with any major effort, not all of the pieces envisioned were finally implemented and tested. Since the

focus of the effort was to lay out a roadmap for a fairly robust system, while focusing on the more immediate problems faced in the simulation facility, a subset of tools was identified early on which expedite development of HUD symbology. More specifically, the critical area identified was the development of software supporting HUD development for the VG system.

Another critical area identified was the need for better standardization and commonality of the software modules developed within the facility. The need for some level of integration for the various existing graphics programs was quite apparent. Based on this, four specific tools were identified, designed, coded, and tested as a direct result of this thesis activity. Additional tools are being researched at this time for inclusion into the project and will be discussed in the concluding chapter.

The four capabilities developed were,

(1) the preliminary buildup of the formalized DESIGNS user interface,

(2) a 'global' executive program which integrates all graphics related software within the facility, and provides the initial portion of the DESIGNS user interface,

(3) a complete definition and implementation for a standard data structure in all display software pertinent to VG hosted symbology, and

(4) an 'auto-build' tool which automatically generates syntactically correct source code for the PACER/VG system from pre-existing symbols.

The description of the DESIGNS user interface, both the hardware components and the dialogue structure, has been given in the preceeding sections. What follows is a complete description of the other items.

The Graphics Executive Program

The graphics executive program functions as the top level of the DESIGNS user interface. In addition, it provides the framework from which ALL graphics related software packages are accessed by the facility users. This package is hosted on the SEL 32/2750, is completely menu driven, and provides the user with a single program from which all other graphics programs can be accessed.

When the graphics executive is invoked from a SEL terminal, the main menu presents the user with the graphics hardware options directly accessible to the SEL. Currently two devices are fully implemented (the two Megateks), and a stub is provided for the Gaertner raster system that is to become part of the system in mid-85. The menu can be extended to include other devices as they are added. The VG is not included at this level, since the SEL has no direct access to it.

The user selects the device which is to be used for the application. The executive software first queries the device to determine its status. If the device is busy, the user is presented with a set of alternatives. The user may elect to

abort the task currently running on the device, designate an alternate device for use, or quit the process entirely.

If the device is available, a second menu is displayed which lists the various graphics programs which are currently implemented for the specified device. Only those programs which can be executed on the selected device are listed so that the user can not inadvertantly select an incompatible program. This menu is presented in a paging structure so that the user may scroll through as many options as are available. The paging permits addition of new graphics programs in the future.

Once an application is selected and activated, the user quits the software option menu and is returned to the initial, hardware selection, menu. From here another device and application can be selected in the same fashion, or the user may elect to exit from the graphics executive. Any or all of the activiated applications can then be used.

This package is used both to initiate the execution of graphics applications and to terminate them. Each menu has been structured to permit easy addition of new packages. Throughout the selection process, errors are tested, and English-type phrases defining them are presented to the user. Appendix G contains detailed requirements for the executive. Appendix H contains formal documentation for the system, including the process by which new applications are added.

DESIGNS itself is viewed as an application program for graphics. Thus it is also accessed through the graphics executive.

Associated with the executive structure resident on the SEL machine, there is a data structure which defines how the simulation programs communicate with the common applications. Several of the Megatek applications are variations on a single display format. Parameter displays can be displayed as,

(1) a 40 parameter display, with two columns of aircraft data plus graphical control buttons for aircraft modes,

(2) a 24 parameter display, with a moving map display, and

(3) a 24 parameter display, with a tracking pipper display.

To promote better commonality among related displays, a single data structure has been defined for passing data through shared memory between the SEL 2750 and the SEL 77's which have the simulation software. This array of data includes control data governing the update of information on the display, all the data describing the function keys, status data, and parameter names and data values. The appropriate subset of this array is used for those parameter displays which contain only 24 entries.

The definition of this data structure was accomplished based on the philosophies which have grown out of this thesis effort. It provides a single structure for those displays that are logically related, which minimizes the integration

that is required on the simulation end.

The VG Database Definition

The definition of a standard structure for the display list pointers maintained by the VG support package has been a key issue in the DESIGNS project. The VG system functions by repeated, rapid execution of a display list which contains coded information specifying the graphics primitives which are to be executed. Within the display list, special pointers may be established so that dynamic updates may be accomplished rapidly, without the necessity for redrawing the entire picture.

These pointers into the display list are 'marked' as the list is being initially built. It is important to understand how the VG display functions to understand the overall importance, and usefulness of the new structure that has been developed for the VG.

The display list itself is an array. As each graphics subroutine call is executed, a command, with all necessary argument values, is placed into that array. As the array is being built, the VG software package tracks the length of the array. All calls to the VG support software require two pointer arguments - an input pointer which specifies the current length of the display list (and thus indicates where in the array the current command will be written), and an output pointer which specifies the length of the array after

the command and its arguments have been added. To maintain the proper integrity of the display list, the output pointer from a graphics routine MUST be the input pointer to the NEXT graphics routine.

If any given command is to be updated in the dynamic loop of the package, the pointer to it within the display list (the input pointer to its subroutine call) must have a unique name. Then it can be individually referenced later in the program. The most straightforward way to preserve these unique pointers is in a FORTRAN array, rather than use many unique variable names. Traditionally, for VG development a dimensioned array, KPNT (400), has been maintained as the pointer array for dynamic updates. This array was contained in a global common so that it could be accessed by all static and dynamic routines.

For clarification of this point, the following excerpt from a VG program is included in Figure 22. This small segment draws a series of lines on the screen, and sets up the system for subsequent translations of the lines. The first pointer argument within any call is an input to the graphics primitive being called and indicates the current position in the display list where new primitives can be drawn. The second pointer argument is an output from the graphics primitive containing the new position in the display list where subsequent primitives will be drawn.

```
                                .
                                .
                                .
call trans  (0, 0, 0, KPNT (1), KOUT)
call disvec (IX1, IY1, IZ1, KOUT, KOUT, IX2, IY2, IZ2)
call disvec (IX3, IY3, IZ3, KOUT, KOUT, IX4, IY4, IZ4)
call disvec (IX5, IY5, IZ5, KOUT, KOUT, IX6, IY6, IZ6)
call trans  (0, 0, 0, KOUT, KPNT (2))
                                .
                                .
                                .
call ?????  ( ........., KPNT (2), KPNT (3))
call ?????  ( ........., KPNT (3), KOUT)
                                .
                                .
                                .
```

FIGURE 22
Excerpt from VG Module, Illustrating Pointer Use

Notice that the initial translate call has a unique input pointer name and the output pointer from that routine, KOUT, is passed to the next routine as its input pointer. This maintains the integrity of the display list. The same name is used for both input and output pointers in these calls. This is normal and syntactically correct. The called modules use the first argument as the starting point to write in the display list, and return the end value in the last one.

In the past, large display packages have been written which used the globally declared KPNT array. These applications contained numerous symbols and could require in excess of 250 of these unique pointers into the display list. Quite often the static generation of all these symbols was written in one long routine or in a series of loosely organized modules with a global common interface so that the pointers could be easily passed from one graphics call to

another.

The problem with this use of the global pointer occurs when the software must be modified. Referring to our very simple example in the figure, suppose the user wishes now to rotate the lines first, followed by the translation. To maintain a unique input pointer for the rotation, there are two options. One would be to use a new variable name such as: call rotate (0.0, 0.0, 0.0, IROTATE, KPNT(1)). If the input pointers are all to be in the global KPNT array then the rotation call becomes: call rotate (0.0, 0.0, 0.0, KPNT (1), KPNT (2)).

With the latter case, which was most often used to keep the pointers in KPNT, all subsequent KPNT entries must be modified since a new KPNT value has been added!! In the example this is not a tremendous amount of work - KPNT (2) becomes KPNT (3), KPNT (3) becomes KPNT (4), etc. In large applications this becomes a significant workload, which is dreadfully prone to errors.

A database structure for maintaining the display list pointers, as well as a standard form for each symbol module was developed to improve VG software development. Under the new methodology, each individual HUD symbol is statically generated within its own module. The unique pointer names are maintained within each module. This permits modifications to any symbol module without updating all successive pointers. The continuity of the display list is maintained by keeping a master array of all the unique

pointers in the global KPNT array. However in the new structure individual routines DO NOT modify KPNT, they modify their own internal pointers. By knowing how many unique pointers each symbol requires, the internal pointers are equivalenced into the proper place in the master array.

A sample symbol module is illustrated in Figure 23. Every module has an input and an output pointer. At the main program level these are passed from one module to the successive module. Looking at this code segment several points can be made.

(1) all modules are identified with a three letter prefix which should indicate what the modules function is (static routines end in 'STC', while the dynamic routine for a symbol ends in 'DYN'),

(2) the same prefix is used for the unique pointers in the module (xxxPNT) as well as the non-unique pointers used to pass in cases where no later updates are required (xxxOUT),

(3) the number of unique symbols required by a given routine is indicated in the dimension statement for xxxPNT,

(4) the first unique pointer for any given symbol is equivalenced into the KPNT array, this equivalence must appear in both the static and dynamic routines for a symbol, and

(5) all the dynamic transformations that could be made to the symbol are accounted for within the original static module (the dynamic module can then change whatever ones are appropriate).

Using this structure isolates almost all modifications to a symbol within the confines of its own static and dynamic routines. It also permits the symbol module to be used for many simulations even though individual simulations may want

```
C     HUD LIBRARY : FIXED AIMING RETICLE          -  STATIC

C     * * * * * * * *   SUBROUTINE AIMSTC * * * * * * * * *
C     *                                                     *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * *

      SUBROUTINE AIMSTC ( AIMIN, AIMOUT )

      COMMON / CNTRL / CON (25)
      COMMON / POINTR / KPNT (400)

      DIMENSION       AIMPNT (2)

      INTEGER         AIMIN, AIMOUT, AIMPNT
      INTEGER         KPNT
      INTEGER         MEDIUM, SOLID, INNER, OUTER


      REAL            HUDSCA, RADIN, RADIUS

      EQUIVALENCE     ( KPNT      (xxx)  ,  AIMPNT (1))
      EQUIVALENCE     ( CON       (  1)  ,     HUDSCA )
      EQUIVALENCE     ( CON       (  9)  ,     MEDIUM )
      EQUIVALENCE     ( CON       ( 10)  ,      SOLID )

      DATA            IXC1, IXC2, IYC1, IYC2  / 0, 0, 0, 0 /
      DATA            RADIUS, RADIN           / 25.0, 1.0 /

      OUTER  =  RADIUS * HUDSCA
      INNER  =  RADIN  * HUDSCA

      AIMPNT (1)  =  AIMIN
      CALL TRANS  ( 0, 0, 0, AIMPNT(1), AIMPNT(2))
      CALL INTEN  ( MEDIUM, AIMPNT(2), AIMOUT )

      CALL CIRCLE ( SOLID, IXC1, IYC1, AIMOUT, AIMOUT )
      CALL CIRCLE ( SOLID, IXC2, IYC2, INNER, AIMOUT, AIMOUT )

      CALL INTEN  ( MEDIUM, AIMOUT, AIMOUT )
      CALL TRANS  ( 0, 0, 0, AIMOUT, AIMOUT )

      RETURN
      END
```

FIGURE 23
Sample Code for a Static VG Symbol

to dynamically drive it differently. One additional change may be required outside the symbol modules. If the number of

unique pointers for a symbol changes, then the equivalence statements for that routine and any subsequent routines must be altered to reflect that change. This change, even if done manually, requires the modification of perhaps a couple dozen lines of code as opposed to potentially a couple hundred.

There are several valid arguments that can be raised regarding the use of the undefined equivalence statement in the modules. There are certainly other ways of implementing this to achieve the same result for the static generation of symbols . The equivalence of KNPT (xxx) with the internal pointer xxxPNT (1) tracks the current 'top' of the display list. The actual value of xxx is determined at the time the modules are linked together into a single source module. This same capability can be achieved by passing the value of the curent display list pointer 'top' as a variable into the module and have the module return the next value. In actuality the xxxIN and xxxOUT pointers do just that in order to properly link modules together.

The equivalence is employed in order to support the eventual dynamic modification of the symbol attributes without having to make all the local pointers common and without having to compute the input pointer when a dynamic change is required. The equivalencing technique provides absolute pointers into the display list array, which is faster than computing the pointer value. Particularly with the PACER/VG system, which is an aging, slow system, the speed issue is a major one when HUDs must run in less than a

100 millisecond cycle time.

The equivalencing technique has been employed, not because it is the best tool in terms of structuring the software, but as a result of considering the FORTRAN IV language issues and the real-time requirements imposed by the antiquated PACER/VG system. In addition the automated modification of the source module by the DESIGNS system is a precusor to more advanced capabilities in which the entire software module will be created and/or modified by the DESIGNS environment. The dynamics of the module can be linked to appropriate simulation variables, which will DEMAND that such source code modification is supported.

This particular issue touches on one of the most controversial areas of the thesis development - that of automatic generation of software. There are strong arguments both for and against having a software system generate another software system. It is the contention of this thesis that it is practical, and necessary for the future of simulation graphics, if such development is to be made with any kind of revolutionary improvements.

The main line program for the VG software has also been standardized so that it contains only the static generation list, the dynamic loop, and the appropriate equivalencing of pointers. From the main line all static modules are called in the form : call xxxSTC (xxxIN, xxxOUT). Again remembering the need to maintain the display list integrity, the output pointer of a routine becomes the input pointer to its

successor. Thus if call yyySTC (yyyIN, yyyOUT) follows the call to xxxSTC there is an equivalence statement for xxxOUT to yyyIN.

This attention to standardization across all the modules was two-fold. First of all, it definitely supported better modularity in the VG software. The structure is therefore easier to modify by a user. In addition, if the structure can be sufficiently defined and implemented in a standard fashion, then automatic tools for modifying it can be designed.

The PACER/VG 'Auto-Build' System

The ultimate goal in developing the data structures for the VG software packages was to provide a consistent system so that an automatic tool could be developed to make required changes when new symbols were linked together to form a HUD. The changes discussed above are very repetitive, and quantifiable. When symbol modules are linked together there are three steps required,

(1) create the mainline equivalences, with the static and dynamic calls,

(2) based on the number of pointers used in each module, compute the proper equivlence point into the KPNT array, and modify the equivalence statement in both static and dynamic routines, and

(3) assemble all the routines into a single file for compilation.

Such a tool was developed as part of the DESIGNS effort. This tool, while initially developed as a stand-alone module, is actually part of the VG definition module within the DESIGNS interpreter. This tool accepts a list of names for the symbols needed in a new display. These names are encoded as the same three letter prefix used in the individual modules. A master table containing the prefixes and the number of unique pointers used by that symbol is maintained.

As the user specified namelist is processed, the master table data is referenced. The files (static and dynamic) for a symbol are retrieved, the equivalence value calculated, and the files linked into a single file. The first symbol specified always has its first unique pointer equivalenced to KPNT (1). Subsequent symbol equivalences are computed. The value is the sum of the current KPNT index plus the number of pointers used in the current module. For example if three symbols were requested, 'xxx', 'yyy', and 'zzz' and module 'xxx' required 4 pointers, module 'yyy' required 7, and module 'zzz' required 2, the following equivalences would be generated -

(1) module 'xxx' - equivalence (xxxPNT , KPNT (1))

(2) module 'yyy' - equivalence (yyyPNT , KPNT (5))

(3) module 'zzz' - equivalence (zzzPNT , KPNT (12))

As a stand-alone tool, the namelist is provided by creating a file containing the three letter prefixes. The master table is maintained as a separate file. Each of these files can be created and modified using the standard editor

capabilities on the SEL system. The namelist will be automatically generated from within DESIGNS in the fully integrated system.

VI. Testing the DESIGNS Concept

The majority of the DESIGNS testing has been accomplished on individual modules. The graphics executive and the PACER/VG data structure have been used as tools within the facility for some time. Various users have had occasion to use them. Both of these tools are principally data structures. There are no sophisticated algorithms which had to be tested for correctness.

The graphics executive software was tested by using the system. A complete walkthrough of all the menus and functions was performed to verify that the system operated according to specifications. Detailed testing was accomplished to prove that the device status could be properly determined. The stub for an additional graphics device was added to verify that new devices could be added to the menu system without encountering major difficulties. The method for accomplishing that was verified and documented.

The PACER/VG data structure has been tested by converting existing VG software to the new structure, and verifying that the resulting HUDs function correctly. New software using this structure has been developed by several users. This use has indicated that development time is greatly improved when the superstructure for every added module already exists. Modification to existing modules has been minimal since the conversion effort. This indicates that, indeed, the design has accounted for the appropriate

symbol dynamics.

In each of these cases the testing has been qualitative. There is no quantitative measure to assess all of the effects these two tools have had. In each case the design has proven to be robust enough to accept changes. Users have been receptive to DESIGNS, with very few negative comments. All of the current simulations within the facility are using both of these new structures successfully.

The auto-build capability for the PACER/VG software has been tested more rigorously. The key areas that were tested include,

(1) verifying that all symbol modules specified are actually retrieved from the symbol library (the special cases for no modules and a single module were tested in addition to an arbitrarily large number of modules),

(2) validating manually that the equivalence values are computed correctly for arbitrarily specified symbols, and

(3) testing the final linked source code file by compiling it and running it on the VG system (each individual module is compiled to test correctness before it is added to the symbol library).

In each case the system performance was determined to be correct. The final test has been to interface an auto-built HUD to a running simulation. This has been successfully accomplished on several occasions.

The most positive aspect of the testing of the auto-build feature has been that it is significantly faster (and certainly more accurate) than building the file by hand. Static displays that can take a couple of hours to create

manually can be created in a matter of minutes using DESIGNS.

The features of the user interface and editor have been, and continue to be tested, in a piecemeal fashion. Preliminary work on the text editor facility was actually done on a micro system. This work served to test some of the menu concepts and the editor specific mechanizations in order to identify the most appropriate method to use in the integrated DESIGNS environment. The micro system was used as all other systems were unavailable at the time. The menu concepts have been transported to the SEL host in a more refined form based on the results of this testing.

Again much of this testing has been qualitative. As has been discussed before, there are no accepted, quantitative measures of the goodness or correctness of a user interface. The menu techniques seem to be acceptable. Some refinements have been made to tune them better for the facility and users. Special attention has been paid particularly to the interfaces between modules of DESIGNS to guarantee the data is passed properly. Typically this has been accomplished by testing and validating the output from a given module, then validating the results the following module produces from that input. Where possible these tests have been performed in such a fashion that the results can be compared with existing graphics software that has been generated in a more traditional way.

VII. Conclusions and Recommendations

Overall the DESIGNS effort has been successful. Certainly the initial plan for the full graphics development environment has not been realized with operational software. However the tools that have been developed have improved productivity, have provided a definitive structure in which to develop software, and have produced an infrastructure under which future software can be integrated. These results are certainly those intended for the development effort.

Conclusions

The major difficulty with the DESIGNS activity has been in scoping the project. As with most theses, the original project definition was entirely too broad to be considered as a single thesis effort. The single most critical failure of the activity has been that the effort was never adequately focused on a specific subject area reasonable for thesis development.

Coupled with the problem of proper scope, was the difficulty of completing the thesis as a part-time student. The demands of other work projects often must take precedence over any thesis activities. This has both prolonged the thesis effort and has made the product less definitive in a sense. With the extended period of time involved, and the work not being performed in a continuous block of time, the focus of the project tended to drift. Many new ideas and

techniques were eventually included that were not part of the original design concept. While this is a very real problem, many of the thoughts that developed over time produced a more in-depth understanding of the task at hand.

As a consequence, both the research and the overall design efforts stretched out and proceeded to cover many areas related to the broad concerns of DESIGNS, its environment, and other closely related issues, but did not necessarily define a manageable thesis topic. Thus, these many sources of information served to keep the scope of the thesis entirely too broad to permit its complete development.

While this sounds particularly negative, the overall effect has been to produce a far better basis of knowlege for the environment definition and for more futuristic extensions. It has made it possible to describe the entire environment design from the standpoint of functional requirements. In the process it has required a great deal of time and yielded but a small set of operational tools.

However, these tools have already had a profound effect on the development capabilities within the sponsoring facility. Each of the capabilities selected for implementation and testing, has indeed proven itself equal to the challenge. They have served to accomplish an original thesis goal - that of enhanced productivity through a set of software developer's aids. It is because of the acceptance and success of this initial tool set, that DESIGNS can be considered a successful project, even though the full DESIGNS

environment has not been completed.

It is apparent from the above discussion that the order selected for the development of the tools has not been optimum from the standpoint of demonstrating the overall system design. However, with facility needs as a primary driver, the order selected has been more immediately responsive to facility demands. The improved development capabilities on the PACER/VG system have made it possible to take two engineers totally untrained in the area of graphics, and have them producing useful software for the system within two weeks. The framework of the top-level graphics executive has reduced the confusion users had when using the system. Before the advent of the executive, users had to remember all the individual file names in order to run programs. Now the single program name is all that is necessary to know. The software is very easy to use, requiring no assistance beyond instructing all users as to the name of the executive program.

It is critical at this point that the DESIGNS effort be continued. Based on the initial success of the somewhat isolated tools, it seems reasonable to conclude that productivity can be further improved with a fully functional DESIGNS environment. Interest in the concept proposed with DESIGNS has been growing tremendously. The ideas and techniques developed with DESIGNS have been accepted as the way to progress into the future of cockpit display design. Even more sophisticated capabilities have been proposed.

Recommendations

In order to provide timely support and information to the organization, the DESIGNS technology should be continually refined and additional capabilities developed for it. The area which should be pursued initially is the completion of the user interface. The full dialogue should be completed to support all of the options from the menus as well as the short-cuts. This effort offers the immediate payback of linking together all of the tools developed to date. Also it is fairly well completed and would require only a small amount of effort to finish it.

In completing the dialogue, the options which are stubbed out, can be expanded. The principal option, that must be expanded, is the graphics editor. At the present time, the graphics editor manipulates libraries of existing symbology. That is a major improvement in the way to do business, but falls far short of what DESIGNS should be able to support. The editor needs to be expanded to support creation of original formats by 'drawing' lines, arcs, conics, alphanumerics, and other graphic primitives. This task should be the precursor to adding the capability to 'paint a picture' with the editor. With the addition of the expanded editor functions more complex, pictorial displays can be supported.

Subsequent to the completion of the full VG system support and editor functions, there are several activities

that could reasonably be pursued. These include,

(1) addition of other target devices, particularly the Gaertner system when it becomes part of the facility,

(2) transitioning all of the text editor capabilities onto the SEL host,

(3) filling out the stubs within the environment to support on-line documentation and dynamic testing,

(4) reconsidering how the symbol libraries for the various systems are actually handled, and

(5) identifying new tools which can continue to support the growing graphics needs, such as the implementation of an image editor.

The symbol library issue mentioned in item 4 is really a key issue. The methodology adapted for this preliminary effort hinged on the fact that the facility had fairly extensive libraries of symbols that are common to a large number of HUD displays. Based on the need to create tools which could off-load some of the workload encountered when creating and modifying HUDs, the structure for the VG software and the auto-build feature were developed. These made use of the existing library of symbols.

However, that library of symbols is specific to the VG. There is also a parallel library that was developed for the Megatek. For future activities it is not reasonable to assume this library will be replicated for each new graphics system that becomes part of the facility. An extension of the internal generic display list to contain a macro description of oft-used symbols offers a better potential solution. The same macro could then be interpreted for each

available target.

It was reasonable at the time to try to use the existing tools within the facility and build on them. However, some consideration should be given to the development of a more generic structure which could require only one library. There are some significant issues to be considered when doing that. These issues revolve around the questions raised earlier concerning the automatic generation of software. Issues such as readability, maintainability, and reliability have been very important with conventional software systems.

Ultimately, these will not be issues in the traditional sense. DESIGNS, or some extension of it, aspires to be a revolutionary new approach to graphics software design and support. In its most sophisticated form it would be capable of far more than has been postulated in this thesis. Ideally, it would not only produce code for displays on any graphics target, but could automatically establish the required dynamic 'hooks' with the simulation software.

In the future if the entire system is so automated, one may not have concerns about issues of readability, because users will never need to look at source code. The term 'source code' as it is used today would be a misnomer. It would not be source in the sense that it is produced today, but WOULD be a type of input source for graphics devices. This type of source software would be maintained and tested within the DESIGNS environment, thus addressing those issues at an entirely different level.

The future of DESIGNS and systems like it are based on accomplishing computing tasks in a fashion unlike those known today. The initial steps taken in DESIGNS have in some ways been constrained in the present due to the facilities which it must support and which are available to support it. The definition of DESIGNS looks toward those future enhancements, and the current interest in DESIGNS-like environments will push for revolutionary new ways to support graphics design.

A serious consideration for future activities with DESIGNS is the choice of language in which the software is written. FORTRAN, even with the nice structures and extensions provided in the 77+ version, is still FORTRAN - with all its woeful shortcomings. Pascal, although more structured than FORTRAN with better support for data structures, falls short in the critical areas of I/O capability and direct interfacing with hardware.

The complexities of the graphics design environment, with its many data structures and interfaces, demand a language rich in data structures and powerful enough to support reliable, maintainable, though intricate interfaces. The best candidate language is Ada. Ada can provide the software facilities which encourage the desired 'revolutionary' approach in the software implementation. When examined, it can be readily seen that the entire DESIGNS definition closely parallels the conceptual design of the Ada Programming Support Environment. The parallelism between the APSE and DESIGNS provides an excellent source of information

for refining the DESIGNS structural definition.

Of equal importance is the nature of the user workstation. All the DESIGNS work was accomplished on a stroke writing system. This is not adequate for the complex work planned in the future with raster and/or CIG systems. A much more powerful CAD type of raster workstation is required to meet future needs.

Future Directions for DESIGNS

Based on the results seen to date, the future of the DESIGNS package seems to be bright. It needs to remain in development, preferably in such a way that the intermediate tools can be used just as the initial tools have been functional during expanding development. It is the intention of the sponsoring facility and the author to continue with the development well after the conclusion of this preliminary thesis effort.

With the assumption that the DESIGNS system is indeed the concept of the future for cockpit design work, it is highly important that the system be reworked in Ada in order to provide the best possible language environment. It would seriously diminish the power of the environment to burden it with a language not capable of serving DESIGNS' lofty goals.

Bibliography

1. ACM SIGGRAPH. Status Report of The Graphics Standards Planning Committee. Vol 13 #3 (August 1979).

2. ACM SIGGRAPH. Computer Graphics - Special GKS Issue. (February 1984).

3. Air Force Flight Dynamics Laboratory. Research and Technology Plan. RCS:SYS-DLX(A)-7402. Wright Patterson AFB, Ohio: AFFDL, August 1981.

4. Air Force Wright Aeronautical Laboratories. Plan for Improvement of Engineering Flight Simulation Capability. AFWAL/FDL Five-Year Plan, preliminary draft. Wright Patterson AFB, Ohio: AFFDL, February 1982.

5. Auld, R., K. Lang and T. Lang. "University Computers Users: Characteristics and Behaviour" in Computing Skills and the User Interface, edited by M. J. Coombs and J. L. Alty. New York: Academic Press, 1981.

6. Aviation Week Staff. "Advanced Simulation Techniques in Use", Aviation Week and Space Technology, Vol 116 (Issue 4): 81-83 (January 25, 1982).

7. Aviation Week Staff. "Digital Avioninc Unaffected by Downtown", Aviation Week and Space Technology, Vol 115 (Issue 45): 181-185 (November 9, 1981).

8. Aviation Week Staff. "European Avionics Systems Advance", Aviation Week and Space Technology, Vol 115 (Issue 45): 191-193 (November 9, 1981).

9. Aviation Week Staff. "Full-Color Cockpit Display Tested by Royal Air Force", Aviation Week and Space Technology, Vol 115 (Issue 45): 185-188 (November 9, 1981).

10. Basehore, Daniel D., Human Factors Engineer. Personal interview on the CADET system design. Air Force Wright Aeronautical Laboratories, Crew Systems Development Branch, Wright-Patterson AFB, Ohio, 15 April, 1983.

11. Bateman, L. F. "An Evolutionary Approach to the Design of Flight Decks for Future Civil Transport Aircraft", Aircraft Engineering, Vol 50 (Issue 593): 4-10 (July 1978).

12. Becker, Rick. On-going exchange of technical data. 1982-83.

13. Boeing. Final Report on Pictorial Format Development to the Gov't, 1 March 1983.

14. Booch, Grady. Software Engineering with Ada. Menlo Park: Benjamin/Cummings Publishing Company, Inc., 1983.

15. Butler, Thomas W. "Computer Response Time & User Performance", 58-62 (December 1983).

16. Curry, Renwick E. and Earl L. Wiener. "Some Human Factors Aspects of Cockpit Automation", The Human-Machine Interface in Airborne Systems. 22-26. IEEE/AESS Symposium, Dayton Ohio, December 1980.

17. Damodaran, L. and K. D. Eason. "Design Procedures for User Involvement and User Support" in Computing Skills and the User Interface, edited by M. J. Coombs and J. L. Alty. New York: Academic Press, 1981.

18. Dudley, Timothy K. "Computers and Graphics: A Technology Comes of Age", Part II, Interactive Computer Graphics Systems, edited by William C. House. New York: Petrocelli Books, Inc., 1982.

19. Eason, K. D. and L. Damodaran. "The Needs of the Commercial User" in Computing Skills and the User Interface, edited by M. J. Coombs and J. L. Alty. New York: Academic Press, 1981.

20. Ewing, D. K. and D. W. Tedd. "A Display System for Processing Engineering Drawings - 'THREAD' - Three Dimensional Editing and Drawing", Advanced Computer Graphics: Economics, Techniques and Applications, edited by R. D. Parslow and R. Elliot Green. 245-259. New York: Plenum Press 1971.

21. Foley, J. D. and A. Van Dam. Fundamentals of Interactive Computer Graphics. Reading, Mass.: Addison-Wesley Publishing Company, 1982.

22. General Dynamics. Display Control Document for the AFTI/F-16 Multipurpose Display Set. 20PP022A, 7 July 1980.

23. General Dynamics. Technical System Description. 16PR2337, pg 8-2. 30 April 1983.

24. Gordon, Mark. "NAPLPS not just for Videotext", Electronic Products, 61-66 (August 15, 1983).

25. Grimes and Ramsey, "How to Design User-Computer Interfaces". Tutorial notes. SIGGRAPH '82 Conference, Boston, Mass., July 27, 1982.

26. Hare, E. W. "COMED - The Cockpit Display of the Future", Aircraft Engineering, Vol 50 (Issue 593): 17-19 (July 1978).

27. Holt, A. P., D. O. Noneaker and L. Walthour, "A Survey of New Technology for Cockpit Applications to 1990's Transport Aircraft Simulators". NASA Contractor Report 159330, Contract NAS1-15546, December 1980.

28. Hubbold, R. J. "TDD: An Interactive Three Dimensional Drawing Program for Graphical Display and Lightpen", Advanced Computer Graphics: Economics, Techniques and Applications, edited by R. D. Parslow and R. Elliot Green. 1035-1047. New York: Plenum Press 1971.

30. Jacob, Robert J. K. "Execatable Specifications for a Human Computer Interface", Human Factors in Computing Systems. Special Issue of SIGCHI Bulletin: 28-34 (December 1983).

31. James, E. B. "The User Interface: How May We Compute?" in Computing Skills and the User Interface, edited by M. J. Coombs and J. L. Alty. New York: Academic Press, 1981.

32. Jauer, J. A. and T. J. Quinn. McDonnel Aircraft Company Final Report. Pictorial Formats, Volume I Format Development. AFWAL-TR-81-3156, Volume I. Wright Patterson AFB, Ohio: AFWAL, February 1981.

33. Kaiser Electronics, Inc. Technical Manual Describing SDS Operation. 1-170. 1983. Internal Tech. Memos defining command structure. 1984a, 1984b.

34. Kupka, I. and N. Wilsing. Conversational Languages. New York: Wiley-Interscience Publication, John Wiley & Son Ltd., 1980.

35. Lewis, James L. "Operator Station Design System: A Computer Aided Design Approach to Work Station Layout", Proceedings of the Human Factors Society - 23rd Annual Meeting. 55-58. 1979.

36. Lieberman, David. "Graphics Standards Bridge Hardware Differences ", Electronic Products, 49-53 (August 15, 1983).

37. Lizza, Gretchen D., Brian Howard and Carole Islam. "MAGIC - Riding the Crest of Technology or Do You Believe in Magic?". Report documenting the MAGIC simulation capability. Flight Dynamics Lab and the BDM Corporation, Dayton, Ohio. 1983.

38. Lockheed. Status report to government regarding Transport Cockpit Design. 1984.

39. Luhrs, Richard A. SOURCLIB: An Interactive Program for Source Code Library Keeping Under MPX 1.4. AFWAL-TM-83-185-FIGD. Wright Patterson AFB, Ohio: AFWAL. August 1983.

40. Martin, James. Design of Man-Computer Dialogues. Englewood Cliffs, N. J.: Pretice-Hall, Inc., 1973.

41. Moss, Richard W., Human Factors Engineering. Series of interviews and briefings on advanced cockpit design. Air Force Wright Aeronautical Laboratories, Crew Systems Development Branch, Wright Patterson AFB, Ohio. Jan-Aug 1984.

42. Mulley, William G. "Future Military Systems", The Human-Machine Interface in Airborne Systems. 14-18. IEEE/AESS Symposium, Dayton, Ohio, December 1980.

43. Mysing, John O. and Michael L. Gravely. "The Role of Digital Technology in Future Cockpit Systems", The Human Machine Interface in Airborne Systems. 37-46. IEEE/AESS Symposium, Dayton, Ohio, December 1980.

44. Newman, William M. and Robert F. Sproull. Principles of Interactive Graphics (Second Edition. New York: McGraw-Hill Book Company, 1979.

45. Norman, Donald A. "Design Principles for Human-Computer Interface", Human Factors in Computing Systems. Special Issue of SIGCHI Bulletin: 1-10 (December 1983).

46. Pavlidis, Theo. Algorithms for Graphics and Imaging Processing. Rockville, Md.: Computer Science Press, 1982.

47. PLAID: Panel Layout Automated Interactive Design, An Overview, Goals, Objectives, and Extensions. Contractor Report. Houston, Tx.: Rothe Development, Inc., 1979.

48. Quinn, T. J. McDonnell Aircraft Company Final Report. Pictorial Formats Literature Review. AFWAL-TR-81-3156, Volume III. Wright Patterson AFB, Ohio: AFWAL, March 1982.

49. Reed, "Device-Independent Graphics Software". Tutorial Notes. SIGGRAPH '82 Conference, Boston, Mass., July 27, 1982.

50. Riesenfeld, Richard F. "Current Trends in Computer Graphics", Computers and Graphics, Vol 3 (Issue 4): 115-122 (1978).

51. Robson, David. "Object-Oriented Software Systems", Byte, Vol 6 (8):74-86 (August 1981).

52. Ropelewski, Robert R. "F-15 Fighter Abilities Evaluated", Aviation Week and Space Technology, Vol 116 (Issue 17): 39-46 (April 26, 1982).

53. Rothe Development. PLAID System Description. 1-8. 1979.

54. Royal Aircraft Establishment. Civil Avionics Research. System description. Dd8294894 Pro. 16587 UK Department on Industry and the Central Office of Information, Her Majesty's Stationary Office by Alpine Press, England, 1981.

55. Schaefer, L. J. "Design of Software and Formats for Interactive Graphics Support", Advanced Computer Graphics: Economics, Techniques and Applications, edited by R. D. Parslow and R. Elliot Green. 803-829. New York: Plenum Press 1971.

56. Sexton, George. Series of phone calls defining cockpit displays for transport aircraft. 1982-84.

57. Sexton, George, Richard Moss and Gregory Barbato. Final Report from the TAWS Simulation. Vol I-III. 1976a, 1976b, 1976c.

58. Sheridan, Susan. "Business Wakes Up to Color Graphics", Optical Spectra: 44-46 (July 1981).

59. Smith, Wayne D. "Beyond the 757-767", The Human-Machine Interface in Airborne Systems. 27-33. IEEE/AESS Symposium, Dayton, Ohio, December 1980.

60. Smiths Industries, product description, 27-33. 1980.

61. Softech. Technical Memo Describing SADT. 9022-78. 4:1-23. 1979.

62. Sutherland, Ivan E. "Sketchpad: A Man Machine Graphical Communication System", (orig. Published 1963), Tutorial and Selected Readings in Interactive Computer Graphics, edited by Herbert Freeman. IEEE Computer Society's Spring COMPCON 80, 25-28. February 1980.

63. Tharp, Alan L. "The Impact of Forth Generation Programming Languages", SIGCSE Bulletin, Vol 16 (No. 2): 37-47 (June 1984).

64. Thomas, R. C. "The Design of an Adaptable Terminal" in Computing Skills and the User Interface, edited by M. J. Coombs and J. L. Alty. New York: Academic Press, 1981.

65. Tucker, Ted W. "How to Organize Process Information for CRT Display", Instrumentation Technology, Vol 28 (Issue 11): 59-62 (November 1981).

66. Tufts. Seminar notes on Fourth Generation Languages. 1984.

67. Wall, R. L., J. L. Tate and M. J. Moss. "Advanced Flight Deck/Crew System Simulator Functional Requirements". NASA Contractor Report 159331, Contract NAS1-15546, December 1980.

68. Warner, James R. and Nicholaus J. Kiefhaber. "Implementing Standard Device-Independent Graphics", Mini-Micro Systems, XV (7): 201-208 (July 1982).

69. Washburn, John and Robert Tibor. "Advanced Cockpit Technology", Technical briefing to government and contractors concerning new cockpit systems. Rockwell International, Collins Divisions, Cedar Rapids, Iowa, June 26, 1979.

70. Weinberg, Victor. Structured Analysis. New York: Yourdan Press, 1979.

71. Whitaker, Richard. "Ferranti Mission Manager's", Flight International: 505-510 (August 15, 1981).

72. Wilson, J. W. and L. F. Bateman. "Human Factors and the Advanced Flight Deck", paper presented at the 32nd International Air Safety Seminar. London, England. October 1979.

73. Wilson, J. W. and R. E. Hillman. "The Advanced Flight Deck", paper presented at the Royal Aeronautical Society Spring Convention. London, England. 16 & 17 May 1979.

74. Vokits, Ronald S. and Harry L. Waruszewski. "Air Force Needs", The Human-Machine Interface in Airborne Systems. 4-8. Dayton, Ohio: IEEE/AESS Symposium, December 1980.

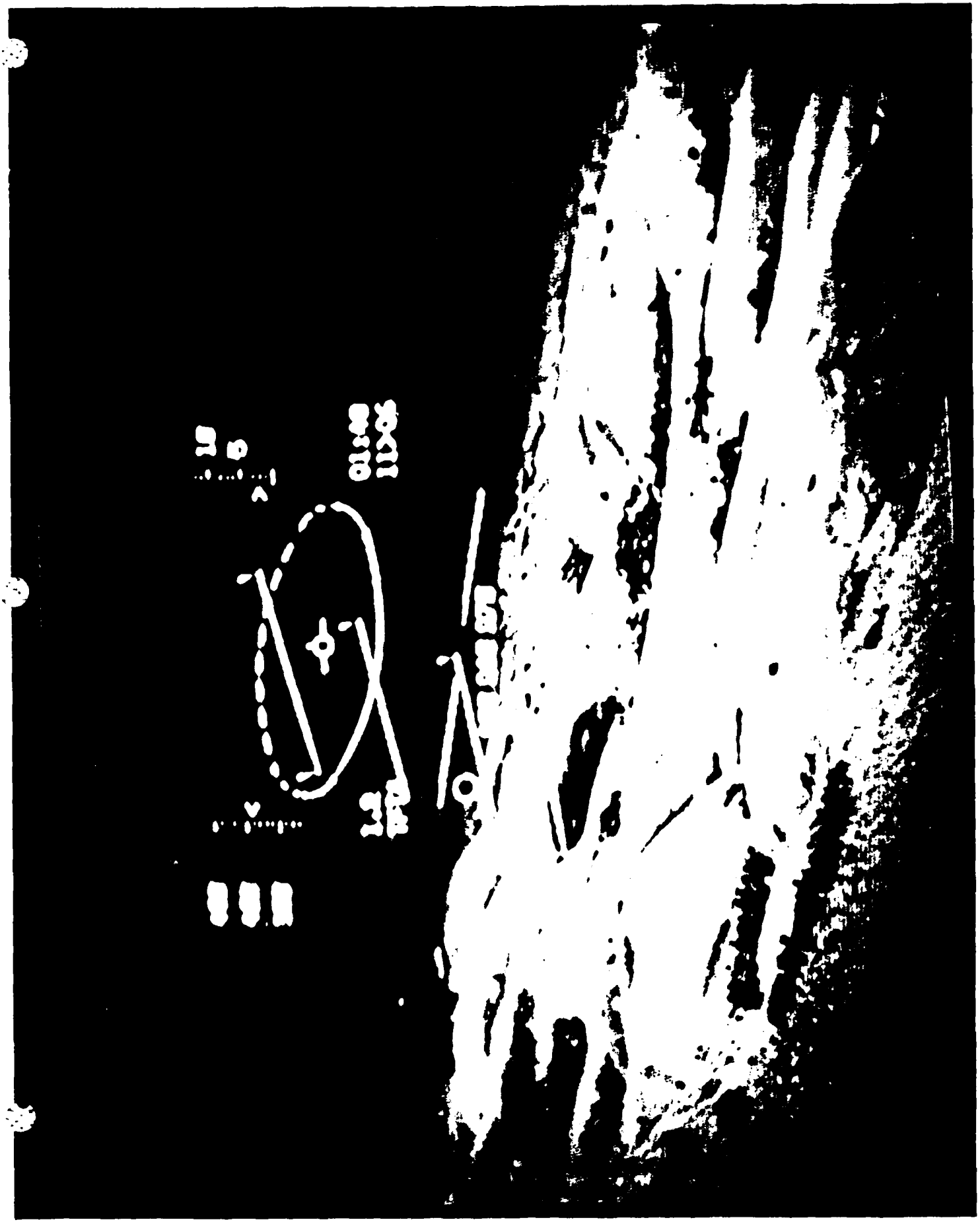
Appendix A. Simulation Topics

Simulation efforts use both heads-up and/or heads-down displays (HUDs or HDDs) in the cockpit. Heads-up formats make heavy use of symbology to encode data for the pilot. The symbols used can be very simple, such as a moving line representing the horizon line reference of the aircraft. They can be extremely complex, such as the pathway in the sky symbol (PITS) which symbolically shows the pilot how to fly a safe profile during the course of an agressive terrain following/terrain avoidance (TF/TA) mission.

This particular type of complex symbol contains many individual pieces of data regarding aircraft attitude, waypoint information, turns, climbs, descents, etc. The HUD in Figure 24 is currently being tested for TF/TA types of maneuvers and illustrates typical HUD symbology.

A certain frame of reference for the consideration of these technical points is necessary to understand how they impact the DESIGNS system. Of paramount importance is the fact that this effort is concerned with research and development (R&D) simulation and its relationship and potential impact on the flying sector of the military. The issues developed here are couched within this framework.

FIGURE 24 - TF/TA HUD Symbology



They reflect the military needs, which in many aspects differ significantly from civilian sector needs. The R&D concerns are of primary importance as opposed to those development needs of the training simulation sector. Again these needs vary from R&D to training installations. All of the following discussions use this frame of reference as the point of departure.

HUD Symbology

HUD symbology is essentially skeletal in nature. It portrays the situation using the bare essentials by providing its data in a symbolic format. HUD symbology willingly trades realism to provide maximum pilot cuing with a stylized symbol set [Boeing, 1983; Lockheed, 1984; Jauer and Quinn, 1982 : 4-13]. Being able to maintain visual contact with the world outside the cockpit, while looking through the HUD symbology, dictates that the display be relatively uncluttered. This leads to the use of symbols to represent important pilot cues. Any alphanumerics used are for digital readouts associated with some symbol. Thus the typical HUD does not use a lot of alphanumeric displays, since the individual pieces of data are encoded within symbolic formatting.

The heads-up display has other common attributes which define it. Because a HUD has been traditionally used in fighter applications or in highly dynamic situations, the

symbology is subject to extensive dynamic transformations at high speeds. This inherent speed requirement has traditionally been met by using calligraphic devices for HUD symbol generation. A second feature is that HUD formats are usually displayed in monochrome only. This is a by-product of using calligraphic systems, since they are primarily monochromatic devices.

It is also important with a HUD to maintain visual contact and proper orientation with the real world. Simultaneously, the pilot must assimilate the HUD data being superimposed over the forward field of view. Therefore, HUD dislays have used the line drawing type of systems which do not block the pilot's forward vision. Symbology composed of lines does not occlude the outside world picture. Raster techniques, while they can provide a more detailed, solid display, can block the view with solid, colored figures and are difficult to registrate with the real world scene. Thus stroke systems have been heavily used for HUDs to the virtual exclusion of the detailed symbology available from raster systems.

HDD Symbology

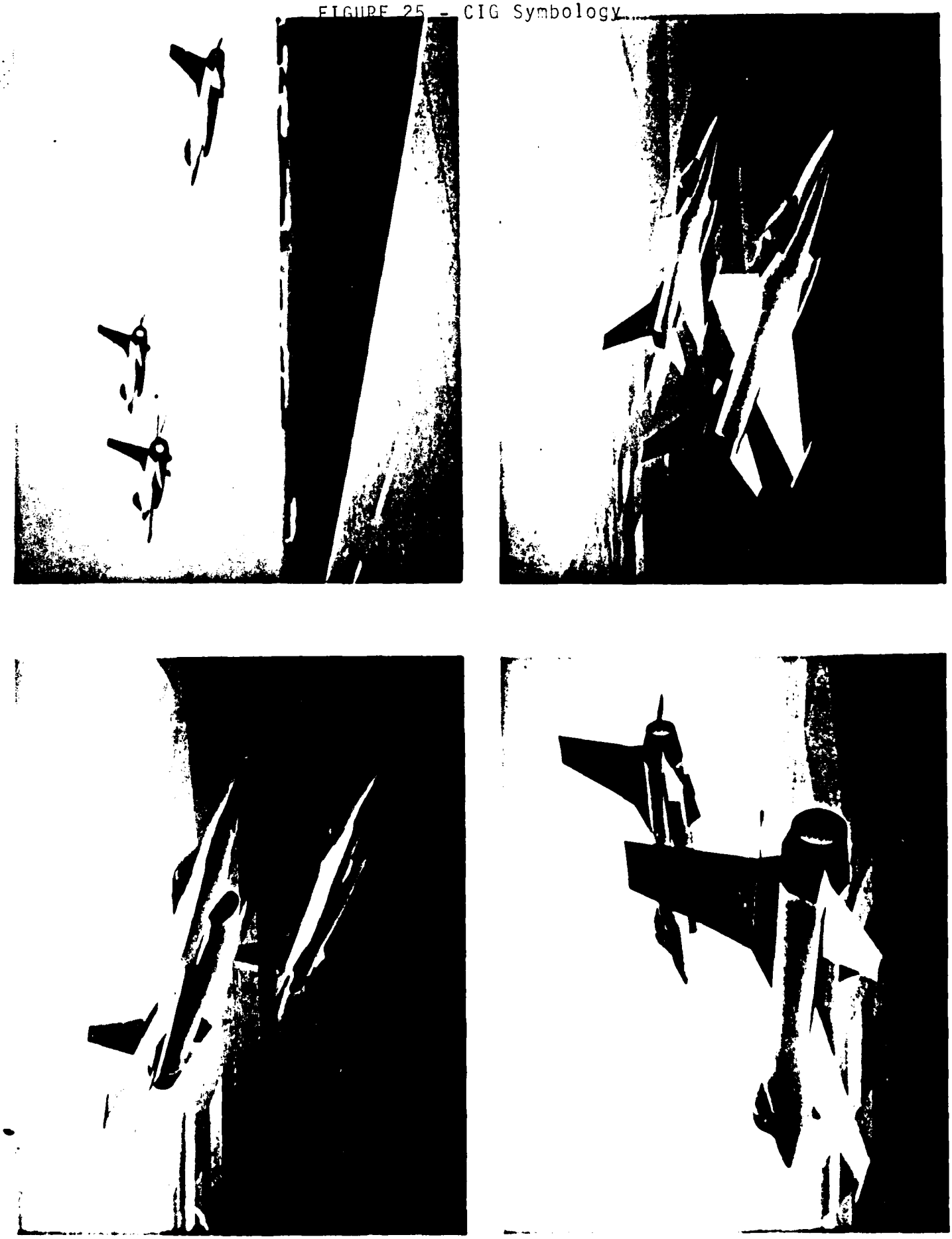
Contact symbology, which provides an artificial recreation of the out-the-cockpit world, is common across the broad range of heads-down display formats. It provides appropriate cues to enable the crew to satisfactorily fly in

visual contact flight. These displays are an abstraction of the real world, as with HUD symbology, and can be highly stylized. The true contact display has come to be termed a pictorial display, as it provides a picture of the situation of interest.

The complexity of some representative heads-down aircraft displays varies from purely alphanumeric to complex pictorial displays. The purely alphanumeric display is the most elementary format with which to deal. Alpha characters and/or numeric data are defined for specified locations on the display. The resulting dynamic operations are limited to updating the contents of the written data. No dynamic transformations such as rotation, translation, or scaling are generally performed. Figure 18, in the requirements chapter, illustrates a typical heads-down configuration for the AFTI-F16 cockpit.

On the other extreme of complexity are the pictorial displays, or image generation. Pictorial formats present data to the crew members in the form of a descriptive picture. The more complex the pictures become, the more computing power and algorithm complexity is required. The furthest extreme of a pictorial format for simulation could present the entire view of the outside world to the pilot. This kind of display would require a large, expensive Computer Image Generation (CIG) system dedicated to performing that function. The extreme detail of an advanced CIG system is shown in Figure 25.

FIGURE 25 - CIG Symbology



While CIG considerations are beyond the scope of this thesis, the pictorial display, in its less complex form, is a likely candidate for DESIGNS support. The intent of the highly stylized pictorials is to produce a 'picture' of the situation of interest. This approach in displays provides a highly integrated description of the data. With well designed formats, the time spent in interpreting and reacting to a given situation can be significantly reduced. This type of format represents the most complex and experimental of the display techniques being investigated for in-cockpit use. This type of pictorial is well illustrated in the series of formats included earlier in Figures 9 through 17.

Somewhere between the all-alphanumeric format and the pictorial design are those requiring some combination of character and symbolic data. These displays typically have provided the mechanism to replace the electromechanical instruments in the cockpit. They are often used to display attitude and situation data. Generating and updating such a display involves the same issues as those for a HUD with symbolic and alpha characters in the format. The Electronic Horizontal Situation Indicator/Electronic Attitude Direction Indicator (EHSI/EADI) combination shown earlier in Figure 3 are typical electronic replacements for older cockpit instruments. They combine alpha data with a picture format, and are actually an electronic copy of the electro-mechanical device they replaced.

Color raster systems are being used more and more for heads- down display generation. Color provides a mechanism to encode data for crew use and to emphasize cautionary or critical conditions. Combined with the color, the raster display provides a powerful tool for the pictorial types of format. The complex symbology can be more easily accomplished using raster techniques. The resulting symbology is more readily recognizable than an equivalent 'open' (stick figure) calligraphic symbology. This reduces the ambiguities and uncertainties within the display. Since the raster representation can closely resemble the actual appearance, these displays are easy to learn and use. The capabilities afforded by color and raster appear at this time to have the best payback in the heads- down display area. Thus, entirely new possibilities are now available for cockpit integration.

Each of the display types described thus far can be used as a part of a simulation project. The heads-up displays, the all alphanumeric, and the symbolic heads-down displays are commonly used. Pictorial formats are in a developmental phase. Their use is increasing in research areas particularly. Research will determine what formats are acceptable for cockpit use. Some contrived pictorials may be of questionable value. More likely a combination of display types and formats will prove to be optimum. Pilot preference and mission requirements will shape the combination of displays that will be acceptable. Simulated scenarios are

being flown using the simple pictorials to determine the feasibility and utility of such formats. For any given effort, the specific project needs dictate the types of display formats to be used. These needs are then constrained or redirected by the capabilities of the facility performing the tests [Boeing, 1983; Jauer and Quinn , 1982; Sexton and others, 1976b; Moss and Barbato, 1980].

Appendix B. Human Factors Topics

Human factors, as related to computer engineering, has been defined as how we humans interact with computer equipment. There are associated areas concerning how we see things, perceptual psychology, and how we learn things, cognitive psychology. The growing body of knowledge in these areas provides a wealth of information that should heavily impact the designs of interfaces between computer systems and the 'world' they service. Traditionally, however, human factors techniques have not been a concern in the formal study of computer systems. [Foley and Van Dam, 1982 : 218-219]

Emphasis within the computer related disciplines has previously been placed on physical computer devices and the software techniques to support them. As computer systems provide more complex services and become available to more groups and individuals, many of whom are not trained in the rigors of computer science, the need for acceptable user interfaces becomes critical. Technical people involved in computer system development are beginning to focus on this problem, both to define it and to provide solutions to it.

The interface of concern in the context of this discussion is one which supports interaction between the computer system hardware and the computer user. Interactive environments should provide support to the system user and make computer usage more simple. Well-defined, powerful

environments should greatly enhance the user's abilities to exploit the machine's facilities.

In discussing the criteria for interactive environments, one important issue has traditionally been the question of system response provided within some tolerable time span. The subjective definition of a good interactive support environment requires that the facilities needed by the user must be implemented and that the environment must not incur delays which exceed the user's patience level. Inadequacies in these two areas break down the dialogue between the user and the system which the interactive system should encourage and exploit [Schaefer, 1971 : 804].

Few people will deny the importance and potential power of a 'good, user-friendly' human/machine interface for a computer based system. The problem with designing and evaluating such interfaces is that there is no definitive measure of 'good' or 'user-friendly'! What is fine for one group or individual may be woefully inadequate or inappropriate for another [Auld and others, 1981 : 78-92; Eason and Damodaran, 1981 : 115-122; Damadaran and Eason, 1981 : 373-387]. These differences among users can be caused for any number of reasons and create a difficult situation for the designer of an interface to a computer system. Human factors techniques often appear to be an art form rather than fully defined theory.

Is this to imply that individual interfaces must always be custom built, tailored to each user's personal tastes? Or that there is no foundation of knowledge on which to start adequate interface designs?

Guidelines for User Interface Design

Fortunately this is not the implication. There exists a solid set of principles and guidelines which can assist the interface designer. Research has provided insight into useful strategies for effective interface definitions. Desired qualities for the dialogue between the user and the computer have been enumerated. Methodologies for profiling the intended user and adequately defining user needs have been proposed. Potential problem areas in the interface development exercise have been identified. System developers can use these guidelines to direct activities in the user/machine interface area.

The quality of the interface plays a major role in the eventual acceptability of any interactive environment. An important measure of this quality is the user's perception of the adequacy of the system interactions. The critical issue is efficiency - and NOT the machine's efficiency, but the user's. An accaptable interactive system must improve the effectiveness of its users and support their activities. It must be a useful tool for accomplishing the user's tasks. Four areas have been identified as critical to the interface

design. They address areas of system

(1) reliability,

(2) adaptability,

(3) self-sufficiency, and

(4) ease of use.

System reliability refers to the system degradation when dealing with invalid user inputs. Carefully designed software interfaces will tolerate improper input formats and provide informative feedback to aid the struggling user. Adaptable systems provide multiple layers of support to present acceptable interfaces for a variety of user skill levels. For example, as a user becomes more proficient in the system use, the level of detail in the interactive messages would decrease. This feature provides an abundance of information to the novice, but doesn't inundate the skilled user with trivia. Self-sufficient systems stand alone in the sense that on-line help is available for users. At any point during the user/machine dialogue, the user may request and receive explanatory information regarding how the system works, what data is expected, or other beneficial items. [James, 1982 : 340, 343-346; Eason and Damodaran, 1981 : 115-118, 120-122]

The ease of use criterion is more difficult to quantify. As discussed earlier, user perception and acceptance of a system varies widely based on the user's background and capabilities. Regardless of the skill levels of the intended user audience, as interactive system should provide the

following capabilities to improve its ease of use :

(1) Small amounts of data, with one major idea per display, should be shown on the screen. There is no justifiable reason for filling a display screen just because it's there.

(2) The displayed formats should be clearly and cleanly designed, with instructions readily discernible.

(3) The design should avoid using difficult words or characters and should provide similarity among the various displays and/or operations.

(4) Required user responses should be kept short and consistant. On-line help should be available and readily accessible.

(5) A mechanism permitting the user to easily backtrack and 'undo' previous operations should exist.

(6) System functions must be clear to the users as well as designers, and must be readily invoked.

(7) Finally, the computer should ALWAYS respond to the user. If a time-consuming function is to be performed, some form of feedback indicating that normal progress is being made is comforting. [Foley and VanDam, 1982 : 218, 222-239]

User or Designer Goals - An Avoidable Conflict

Adherence to the above guidelines does not necessarily guarantee the resulting system design will be acceptable. The inherent difficulty lies in the great disparity that often exists between the system designers and the system users. The problems and interests of the system programmer have often been confused with those of the eventual end-user. Developers many times consider themselves as typical users, thus making their own system design objectives the focus of attention. The requirements of the final application use

have been relegated to an inferior position, or ignored, which potentially compromises the design. This approach will surely design a system that is a programmer's delight, but likely a user's nightmare. This failure to recognize that program designers are, in general, NOT typical users, results in software packages that may have been very clean to design and implement. They may exploit the physical machine's capability to the limit. But they likely provide a weak, awkward interface for the ultimate user. [Foley and Van Dam, 1982 : 217-8; Grimes and Ramsey, 1982 : 2-3 to 2-4, 3-4 to 3-7; James, 1982 : 345-46]

Therefore, it is important that the system designer consider seriously the end-user needs and capabilities. The design focus shifts radically when it is truly based on the end user requirements. Much more time is devoted to tailoring the system for the end user. User tailoring places the focus in the proper perspective - with the attention directed toward the abilities and needs of the user.

By and large the profile of users of computer based facilities can be characterized as [Martin, 1973: 19-23]:

(1) intelligent,

(2) too busy for in-depth training,

(3) highly impatient,

(4) non-rugged, and

(5) demanding of worth while results.

In considering ourselves as users, most items above can be readily accepted as reasonable attributes for a computer user. We certainly like to consider ourselves as intelligent. As such, we do demand results which are timely and worthy of our consideration. For the most part we do not intend to devote vast amounts of time to learn any given system in detail, just well enough to get our job done.

Combinations of these traits cause the user to be described as non-rugged. Non-rugged in the sense that as a reasoning being, a user has certain expectations of a system. Such a user will not willingly tolerate system behavior significantly inferior to those expectations. System response time, how quickly can it accomplish a task, is a critical point of user evaluation. A sluggish system is intolerable to most users. Ironically, the same user is equally unaccepting of a system whose response time is too rapid. This can intimidate particularly the casual user who may feel pushed to respond much faster than seems comfortable. People need to break work into sections to attain a feeling of closure on one activity before proceeding to another. System response which is so fast as to preclude such breaks is stressful and ultimately unacceptable. An elusive balance must be established between the user expectancy of a smooth response time and the user's need to perform work in segments with some break between them. Time estimates for how quickly a system should respond and how long breaks should be vary considerably with different users

and applications [Grimes and Ramsey, 1982 : 3-2 to 3-7; Martin, 1973 : 10-13, 19-23] .

The discussion above presents several assertions that are accepted as common knowledge regarding the effects of response time on human performance. Interestingly, there is very little empirical data to either support or refute these theories quantitatively. One recent study refutes the theory that inappropriate response time impairs performance [Butler, 1983 : 58-62]. Butler finds poor statistical correlation between response time and performance in a variety of tasks. However, Butler's experiments do not consider the user's acceptance of the system's performance. He merely quantifies the number of typing errors made during a session and determines that the error rate is not statistically effected as the response of the system degrades.

Other studies [Foley and Van Dam, 1982 : 217-218, 239-242; Grimes and Ramsey, 1982 : 3-4 to 3-6, B-5, C-4], which include subjective comments, conclude that the user acceptance is affected by the response time of the system under question. Ultimately, the user's perception of the system is a critical factor in satisfactorily performing tasks over a long period of time. Therefore, the issue of response time constraints will be carefully considered in order to assure user satisfaction.

User/Machine Dialogue Characteristics

Traditionally, two distinct techniques have been used to establish a dialogue between the user and a computer system. Systems can be classified according to the mechanization of the dialogue as either menu driven or command language based. The choice of dialogue type must be made with a clear understanding of both the system requirements and the technical merits of the menu and command language implementations.

Before discussing the technical points of either dialogue type, it is necessary to define what is implied by each alternative. A menu driven system is one in which all valid selection options are displayed to the user in some form of menu. Only those options currently displayed by the menu may be activated from that particular menu. Selections can lead to other menus or to the performance of an activity. Thus to access other options, other menus must be selected first. Selections from the menu are made by pointing, in some fashion, to a menu item or by typing a short command indicated by the menu. In either case the method for selection is available on the display screen.

A command language system presents no such visual aids to the user. The system options are selected by typing the command for the desired action in the syntax required by the operating system. All of the valid options in the entire system are available at all times for activation. The user is required to know the required syntax for actual invocation.

These definitions are necessarily restrictive in nature as they make no attempt to address dialogue types that embrace concepts from each of these techniques. Nor do they address some of the differing techniques that have been attempted to provide different dialogue forms. These two techniques are both popular and successful, and will be used as the basis for the following discussion. This does not preclude the consideration of other related techniques if indeed they can be better suited to the DESIGNS effort.

In addition to the command and menu driven techniques, there are numerous other dialogue implementations. Many of these are the result of combining the styles from command language systems with those of menu-driven systems. Each has its own distinct set of shortcomings to accompany its strengths.

This section contains information for other techniques with which a human/machine dialogue can be implemented [Grimes and Ramsey, 1982 : 7-2 to 7-4; Martin, 1973 : 12-13]. This shoulf provide the reader with a more complete picture of the dialogue types available. Keep in mind that each of these to a certain extent is merely a permutation or extension of the command or menu-driven techniques.

Dialogues can be classified in terms of the type of conversation that is being handled. There are two types - that which the operator/user initiates and that which the computer initiates. Those conversations which the user initiates may seem to be more under the user's control, but

also require the user to be more knowledgable. Computer initiated conversations can be intimidating and confusing if the informational content is too high, or if presented to quickly.

The following techniques are typical user initiated conversations. The list is ordered from simple to complex. In each case the user types in a command as in a traditional command language system. As the complexity increases, the form of the command is more involved, and multiple actions can be simultaneously initiated. In the most complex commands, a series of actions are commanded from the keyboard. The user initiated conversations include:

(1) simple query, which is a single, simple command,

(2) mnemonic techniques, which provide a shorthand, encoded form of a command convenient for frequent users, but cryptic in actual form,

(3) English language techniques, which provide a more natural mode of communication,

(4) programming statements, which resemble computer language statements and may initiate complex activities with a single command,

(5) action code statements, encoding desired system actions into recognizable commands,

(6) and multiple action code systems, which initiate multiple activities with a single command.

The computer initiated conversations can assist the user. Many of these techniques are based loosely on the menu-driven system characteristics discussed in the body of this paper. The more complex implementations can present a high workload situation to the unfamiliar user. The

following list presents some of the more common approaches to computer initiated conversations. As with the first list, these are listed in order from most simple to most complex. Examples of computer initiated dialogues include:

(1) instructions to the operator/user, which request user input, describe legal actions, and require some user response,

(2) menu selections, which may include simple menus or multiscreen, nested menus, or menus requiring multiple answers,

(3) encoding information in formats, which presents more of a picture to the user,

(4) variable length and multiple format entries, which permit the user to select default options on parameters when acceptable,

(5) form-filling,

(6) text editing techniques,

(7) overwriting, which permits the system to overwrite information within a selection and can be extremely confusing,

(8) and, as a kind of catch-all, hybrid dialogues, which provides the designer which great latitude in the dialogue development to pick and choose those techniques which may be most beneficial. The only problem with this approach is that consistency can often be lost.

When considering the computer initiated conversations, one must also consider the way in which information can be encoded for presentation to the user. One way to assist the user, to offset the more limited bandwidth capacity of the user as compared to the computer system, is to use displays of some type. The use of displays also aids the cognitive process and supports the user's short term memory limitations by not requiring memorization of lengthy commands.

The following table illustrates the value of various drawing techniques. Of particular importance is the very positive effect of using recognizable, geometric shapes. This technique plays an important role in the DESIGNS menu system. It was selected because of the benefit to the user.

Selection of the Dialogue Type

The differences between menu and command dialogues tend to make each of them appropriate for distinctly different audiences. Menu driven dialogues, with the ever-present selection aids on the screen, tend to be well suited for casual and infrequent users. Since there is no requirement to memorize long command sequences, such a user can work through the entire system using the menus fairly easily. There is an accompanying penalty with this system in that the menus take a longer period of processor time to draw on the screen, but casual users seem not to adversely suffer from the somewhat decreased response time. The additional help available more than offsets the writing time required for the menus.

The highly trained user finds the menus to be a hindrance. The design philosophy being one of providing help whether the user wants it or not, precludes instant selection of all system options. Typically the expert user does not want, or need, all this help. This type of user much prefers the flexibility and speed of the command language system to

TABLE VI
Effects of Display Encoding Techniques

ENCODING METHOD	RECOMMENDED MAX NUMBER OF ITEMS	RECOMMENDED MAX FOR RAPID/ERROR FREE REACTION	COMMENTS
Numerals, and letters	Unlimited	Unlimited	Highly versatile, little aid to memory. Location time can be long.
Geometric shapes	approx. 15	approx. 10	Very effective, high mnemonic value. Supports the cognitive process.
Angle of Line	16	8	Good in some cases
Line Length	4	3	Fair, clutters displays
Line Width	3	2	Good
Solid Lines	9	5	Good
Object Size	5	3	Fair, requires space and longer location time than for shape or color
Brightness	4	2	Poor, especially if the screen contrast is poor
Flashing Symbol	4	2	Poor, confusing. But can attract attention easily.
Color	11	6	Expensive, valuable. No extra space required. Location time short. Can relieve clutter.

the abundance of information available with menus. The user expert in a system knows the command structure well enough to select desired options and often knows shortcuts to accomplish tasks. Menus only serve to bog down the efforts of such a user by strictly structuring how all tasks are accomplished.

There are other differences that impact the user beyond the differing physical apperances and the selection methods of the two dialogues. The selection mechanism used in the command language implementation is somewhat impervious to undetectable user errors. If the command syntax is incorrect, the system will simply reject it. Improper actions are, in general, immediately apparent. In a menu driven system, errors can be committed in more subtle ways. Since all the displayed options are valid for the system, an invalid selection cannot be made. However, while a selection is perhaps valid, it is not necessarily the correct one for the users intentions. Particularly if the selection leads to deeper levels of menus which are common to, or similar to, ones for several options, the user can quickly and easily become buried doing an unintended activity. As menus levels are traversed it becomes more difficult to undo the steps that have been accomplished.

Serious errors of commission can occur in either system by careless operation. The severity of the errors in a menu driven system can be greatly reduced by competent dialogue design. But these are some of the major points concerning

the dialogues. Table VI contrasts the more of the positive and negative aspects of the two dialogue designs. [Foley and Van Dam, 1982 : 218-219; Norman : 1983 : 2-6, 9]

TABLE VII

Menu Driven vs Command Language Based Dialogues

ATTRIBUTE	MENU DRIVEN	COMMAND LANGUAGE
Speed of use	Slow, especially when menus are large or there is a hierarchy of menus	Fast, for experts; Operation specified exactly, regardless of system state
Prior knowledge required	Self-explanatory; minimal previous information is necessary	Significant knowledge about actions and command syntax is necessary
Ease of learning	Quite straightforward; Requires recognition memory which is much easier than recall; Easy to explore system and locate available options	Difficult; User must memorize syntax of language which if extensive requires much time; No easy way to find an option not known
Errors	Specification errors lead to inappropriate actions that are both difficult to detect and correct	Specification error usually leads to illegal commands; These are easy to detect/correct (if syntax is known)
Target User	Beginner or infrequent	Expert or frequent

From the discussion and the comparison data presented in

Table VII, it can be seen that each dialogue technique has its positive as well as its negative aspects. Consequently, there is no universally best system for an actual implementation. There are only tradeoff issues to be evaluated in light of the system requirements.

Appendix C. Hardware/Software Integration Topics

The DESIGNS concept is heavily dependent on a thorough understanding of both software principles and graphics hardware capabilities. The defined environment is somewhat hardware intensive since the key element involves the targeting of software to multiple graphics devices. In order to manage effectively this targeting of software to a family of dissimilar graphics systems, it is important to understand the kind of capabilities and support that can typically be found in the graphics systems.

With the understanding of the hardware functions and the software techniques available to model the system definitions, an integrated environment can be designed. Thus, the nature of the DESIGNS concept requires an integrated look at the hardware and software issues because of their close inter-dependency.

There are certainly topics within the hardware and software areas that can be discussed as separate issues. Several of these individual issues have been identified as critical and their importance to the DESIGNS definition or implementation recognized. These issues form the basis for the DESIGNS concept and involve many areas from both software development and hardware interfacing. In addition, there are those topics that bridge the traditional gap between pure hardware and software issues. These areas are primarily of concern with the issues of interfacing the DESIGNS functions

to the various target devices. It is this topic, whose successful completion is critical to the success of the DESIGNS concept, that must be considered from the system integration point of view.

The software required to support the DESIGNS concepts is heavily involved with database definition and implementation, abstract data type definitions, dialogue support, and algorithm development. Hardware specific considerations impact the definition of critical system interfaces and seriously affect the software which must support these interfaces.

Overview of Critical Hardware Issues

The choice, availability, and capability of the hardware devices used have far-reaching consequences on the overall DESIGNS effort. While it is desirable to maintain portability in the DESIGNS package to facilitate rehosting if necessary or desired, it must be recognized that at some level the idiosyncrasies of the host hardware as well as the target system hardware must be addressed. It is important to identify areas in which these hardware issues impact the DESIGNS package, so that hardware specific software can be isolated from the remainder of the package and provide a better mechanism for device independence.

The following sections discuss the features that are potentially available within the DESIGNS hardware systems.

Typical capabilities of today's workstation equipment and graphics devices are described. Tradeoff issues, along with supporting data from previous research, are discussed.

Typical Workstation Capabilities

A user workstation can provide support input/output (I/O) functions ranging from simple, austere capabilities to very elaborate support facilities. While discussing workstation issues, areas that must be considered are system input/output functions, display screen capabilities, special features to support graphics, and general issues such as error handling, system security, reliability, portability, maintainability, and flexibility. [ACM SIGGRAPH, 1984 : 38-39; ACM SIGGRAPH, 1979 : V-2 to V-7; Foley and Van Dam, 1982 : 18-27, 95-135, 183-197; Martin, 1973 : 19-23; Newman and Sproull, 1979 : 3-8].

Most of the workstation issues are important by virtue of the impact they have on the user interface. The workstation is the only physical interface that a user has with the DESIGNS environment. User critical areas have been discussed at some length under the human factors topic.

There are many input devices that are fairly standard equipment for a broad range of modern graphics systems. More systems are being designed for the explicit purpose of being stand-alone workstations. These stand-alone systems usually support multiple input and output devices that are useful for

a variety of graphics related tasks. The majority of the systems can support such devices as joysticks, mouse devices, data tablet, light pen, control knobs, valuators, and a variety of other technologies. There are more sophisticated devices also available. These are not really necessary to accomplish the functions required for DESIGNS.

The DESIGNS capabilities discussed in earlier chapters can be accomplished with relatively simple devices. For example, the designation of symbols can be accomplished by such things as light pens, or any device which can move a cursor (such as trackball, mouse, joystick) and activate a selection (event button). The drawing of new symbology likewise can be supported by a device such as a digitizing table.

There are detailed display screen issues to be considered. A determination of the amount of symbology that must be displayed at any one time during a DESIGNS session is necessary so that the physical display capacity can be specified. For raster type displays this implies a definition of the raster size, or resolution, necessary. For stroke, or calligraphics systems, the number of vectors supported is necessary. Drawing speed, update rate, and refresh rate can be critical parameters for various applications. A very basic issue to address involves the use of a raster versus a stroke generated display. There are significant tradeoffs between the two as discussed earlier in this paper under the cockpit designs topic. The choice of

the workstation display reflects these considerations and effects the use of the display screen.

In addition to the I/O issues, Martin and Van Dam caution that there are other issues, less likely to be considered, that may have a serious impact on the choice of system hardware and on later implementation. One must consider early in the design phase the necessity for the workstation to support special graphic functions. The major issues here concern the need for special capabilities such as surface filling and shading, special high-speed options for transformations, any advanced algorithms, color options, and many other of the available graphics options. Associated with selecting these capabilities or not, is the concern as to whether or not some of them can be supported by software algorithms, or if they require dedicated hardware to provide faster speeds.

In another area, system security concerns may have long term impacts especially in a military environment such as is postulated for the DESIGNS system. Certainly some level of security must exist in order to maintain the integrity of the DESIGNS database th t supports the project. Likewise, there are long term ramifications in the areas of reliability, maintainability, and flexibility. Ease of servicing and upgrading the workstation may be important considerations of the unit is to be used as a tool for an extended period of time.

The workstation is a physical plant, and as such has the same concerns as any other physical installation. Many of the issues may seem to border on the consideration of minutiae, but they are valid concerns over the lifetime of the equipment. It may be important, or mandated, that the system meet specified noise levels, or size constraints, or particular compatibility with existing pieces of hardware.

The details of these areas discussed in rather broad terms can not be ignored. At least one must make the conscious decision that the impact in some area is mininal and thus is not an important concern.

Target Device Interfaces

Even in a fairly restricted operational environment, it is likely that multiple graphics processors will be available for use. In all likelihood, these devices will be dissimilar both in their hardware characteristics and system software support. The intent with DESIGNS is to provide for target device independence, at least from the user's perspective. [ACM SIGGRAPH, 1984 : 201-207; Reed, 1982 : 281-289].

The necessary knowledge for delineating such a generic definition lies in two areas. First, an understanding of what hardware options are available, and thus must be made accountable/accessable through the definition, is necessary. Second, an appreciation for the software data abstraction methodologies is paramount to the successful implementation

of the standardized definition.

The importance of device independence and graphics standards is widely recognized within the graphics community. Substantial effort has been taken to define the functional capabilities supporting device independence [ACM SIGGRAPH, 1979; ACM SIGGRAPH 1984]. Such standardization supports quicker, easier, less costly graphics development, while also providing for portability of both software packages and personnel.

Both the CORE Standard and the Graphical Kernel System (GKS) define requirements for predefined capabilities within the basic software package. There is also some attempt to define the requirements for a virtual device interface (VDI) which provides for standardization of hardware. This is the weakest point in both CORE and GKS specification, with only a very generalized statement regarding the VDI. Neverthless, the initial steps have been taken in an effort to define and promote graphics standards. These standards have been embraced by the American National Standards Institute (ANSI) and the graphics industry. Continued development of standards, with full implementations of them, is being actively pursued by both groups [Lieberman, 1983: 49-51].

These efforts provide a precedent for the DESIGNS effort. Concepts from the standards provide a basis for the definition of the environment capabilities and characteristics.

Selection of Host Computers

The selection of a host computer can have far-reaching consequences on the overall DESIGNS effort, but possibly none are as critical as the effect the host could have on the actual development effort. The intent for the DESIGNS environment is that it could be used in several host machines if so desired. The system design is not necessarily tied to any given type of host hardware as long as the features specified for graphics support and data storage are available.

Thus the critical impact of the original host machine is its capability and availability to support the development effort for the thesis. Strong development support is necessary for the successful completion of the implementation phase of DESIGNS. The availability of support tools such as programming languages, editor support, and file management resources is imperative. A consideration of the available tools within the host operating system and supporting subsystems is important when considering a potential host.

Equally important is the host availability. If potential host machine resources are already heavily committed, then the development time for a major effort such as DESIGNS becomes prolonged. It may be more effective to select a host with lesser, but adequate, capabilities which has better availability.

There are no unique features required for the DESIGNS development effort. The host selection need only consider the availability of the host, its resources for the development of the software, and the utility of its support functions. The important characteristic of the host, since most computers today offer the kinds of features required, is that the features be reasonably simple to use.

Overview of Critical Software Issues

Many of the software issues have been alluded to in the simulation, human factors, and hardware topic discussions. The actual software implementation must, of course, reflect the needs and constraints in each of these areas. Ideally, it is this software package that integrates all of the desired functions, but in such a manner as to minimize any of the potential shortcomings. Earlier technical discussions serve to identify key areas for software development.

The DESIGNS implementation is heavily dependent on database design and data abstraction techniques for the storage of symbol definitions, generic display list constructions, and target machine descriptions. Several research efforts have proposed data structures which can be manipulated by graphics tools. The work of the Graphics Standards Planning Committee of ACM produced the CORE Standard package late in the 1970's. This was an involved effort to produce a true standard for graphics, particularly

graphics software. The GKS proposed standard, which evolved later, addresses the area of data structures which should define a graphics environment. Other de facto standards, such as the North American Presentation Level Protocol Syntax (NAPLPS), are finding acceptance within the graphics design community.

Each of these standards provides a basis for definition of various critical data structures or communication paths which must be embedded within the DESIGNS environment [ACM SIGGRAPH 1979; ACM SIGGRAPH 1984; Lieberman, 1983: 49-53; Gordon, 1983: 61,64]. Unfortunately, there is no clear-cut standard at the moment in the commercial market, even though the GKS has been adopted by ANSI as a standard. Graphics vendors, both hardware and software, are hesitant to commit one way or the other at this point due to the tremendous cost of an incorrect choice.

Another area of software development involves the DESIGNS dialogue. Much attention has been devoted to the definition and description of characteristics of various dialogue types. The implications of these discussions on the actual software implementation are many. Numerous studies have been conducted to provide clearer guidelines for dialogue implementations to assure that proper software techniques are employed. Parsing techniques provide the necessary capabiity to interpret dialogue activity.

A tremendous amount of discussion is on-going within the computer community today regarding the meanings, benefits, and applications of fourth generation languages. How do these 'new' language capabilities impact the DESIGNS effort, if they do at all? Some concepts from the fourth generation language proponents bear interesting relationships to the proposed DESIGNS concepts. Thus the issues surrounding fourth generation languages become, in some sense, part of the software issues of DESIGNS.

All of these software issues ultimately revolve around the final design of the algorithms necessary to accomplish the DESIGNS objectives. The DESIGNS development has been predicated on the available technology base and theoretical techniques. To fully understand the detailed definition of the DESIGNS concept, one necessarily must have the relavant background information. The following sections expand on the topic areas enumerated here and set the stage for the actual system design description in the next chapter.

Graphical Data Structures

The need for data structures to define graphical objects has been understood from the early works of Sutherland with the SketchPad dissertation. As experience with graphics and computers broadened, the refinement of those data structures used for graphics began. However, many of the characteristics of the data structures in common use today

are very similar to the first definition implemented by Ivan Sutherland in the early 1960's.

The Sutherland database dealt not only with the symbology itself, but also with those constraints necessary to fully define it. His database mechanism recognized that in drawing symbology there were constraints that must be satisfied. These constraints were the basic relationships that made lines straight, or vertical , or parallel, etc. There exist both geometric constraints and user imposed constraints.

The geometric constraints are more easily quantifiable as they represent some definition for drawing a given symbol. For example, a square can be drawn in one of several ways - by specifying the four corners, by specifying one corner with the length of the side, etc. Each of the definitions for drawing a square is complete in and of itself, but contains a differing set of constraints.

Sutherland based the generation of symbols on a definition copying scheme. Within the SketchPad system existed definitions of different ways to mechanize a symbol. From that definition, new symbols were instantiated by 'copying' the definition with the proper points, lengths, etc. supplied. This technique is still used today, in a refined form within the Smalltalk environment. This environment, developed as a research tool for defining advanced, graphically oriented systems, is an object oriented system which operates by using definitions which describe how

something is to be accomplished.

Sutherland's data structure implemented rings of information. In this scheme the general information is separated from the specific by collecting all things of a given type under a generic heading. Thus the unverse of symbols can be defined in terms of variables, constraints, topologies, and holders (which are attributes of multi-symbol segments) [Sutherland, 1963: 25-28].

Outgrowths of the SketchPad system include two systems used for engineering related graphics. Each of these also is based on the ring data structure. The first, a Three Dimensional Editing and Drawing package, THREAD, uses what it terms an Associative Structure Package, ASP. Within the ASP, individual objects are defined by linking rings of data which are associated by types. Figure 26 illustrates this data structure for some generic object [Parslow and Green, 1971: 245-259].

The second system, another Three Dimensional Drawing system called TDD, is an interactive graphics system. This system uses a more complex data structure composed of rings and beads, a bead being a block of data containing both problem data and pointers to other related parts of the overall graphics data structure. In TDD all information is stored in linked tree structures by defining object edges and vertices. The edge information is related together in a linked ring. This particular data structure becomes very large, and is difficult to modify rapidly as the pointers

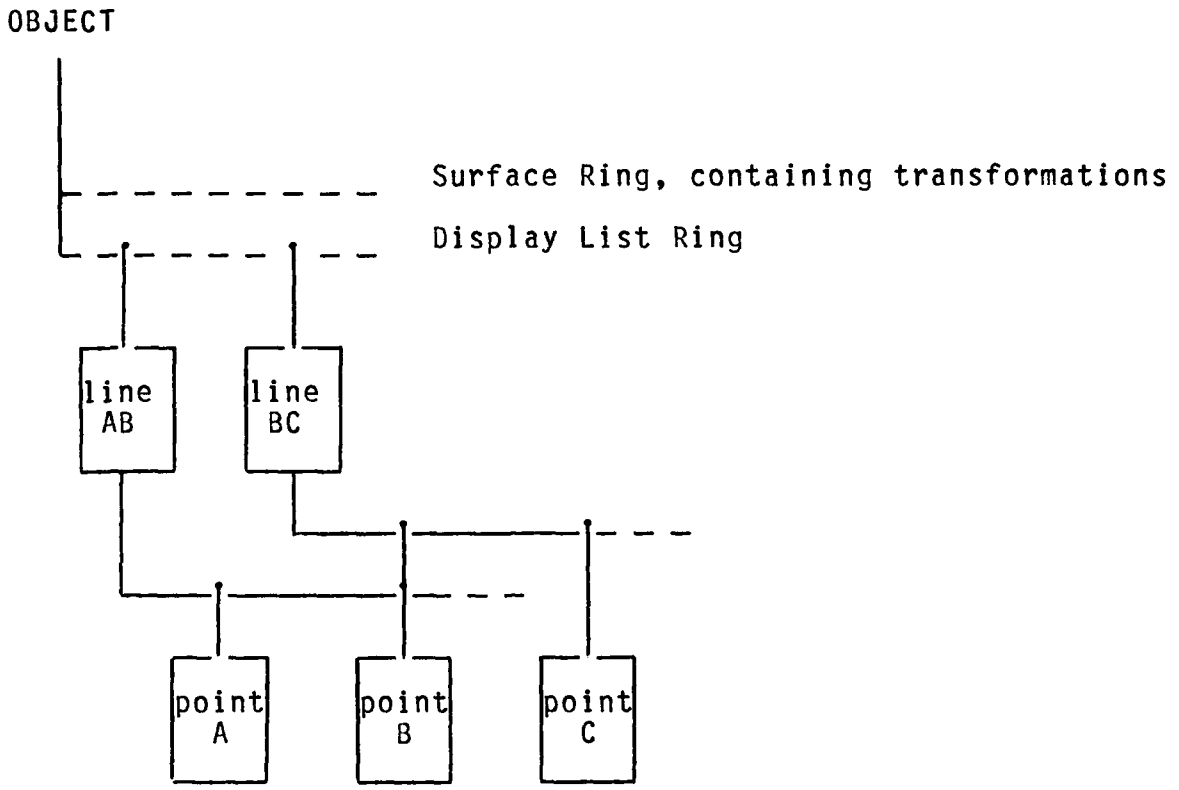


FIGURE 26
ASP Ring Data Structure Implementation

must be traversed. However, it is excellent for an interactive environment. With the dynamic data allocation supported by the addition of pointers, this data structure lends itself very well to activities encountered in the interactive systems.

As work has progressed into the 70's and 80's, the data structuring problem has been recognized as acute in graphics applications. In order to facilitate more efficient, smaller data space requirements, relational databases have been

studied. Internal tables of data defining graphical attributes have been used in various picture building systems. The relations are then interpreted by a relations editor and symbology is generated.

Accompanying this move to relational database structures is the deeper understanding that for graphical data to possess any meaning it must have both syntactic as well as semantic meaning. The mechanism of how the data is stored and accessed, its syntax, is only part of the story. How the data is to be used, its semantics, must also be understood.

Traditionally, the semantic information is not stored and can not be operated on. Thus programs and data were inseparable as the semantics are buried within the program calls. Newer graphical data base methodologies are advocating the inclusion of graphical semantics. Table VII below is an excerpt showing the type of semantics defined for an early system using this approach.

Graphics Standards

Serious efforts in the late 1970's and 1980's have been undertaken to standardize graphics software. The CORE Standard evolved as practicing graphics developer's recognized the need for a defined, standardized methodology. The singular driver for the development of the standard was the issue of portability, both of software and the people who develop and use it. The CORE definitions focus on specifying

TABLE VIII

Semantics Permitted in Early Picture Building Systems

O P E R A T I O N	S E M A N T I C M E A N I N G
Rx, Ry, Rz	Rotation around specified axis
Scale	Scale equally in x, y, and z
Scale x, y, z	Scale in the specified axis
Shift x, y, z	Shift (translate) in the specified axis
WLLX, WLLY, WURX, WURY	Corners for a window - lower,left x and y upper,right x and y
VLLX, VLLY, VURX, VURY	Corners for a viewport
Line	Draw line to x, y, z
Move	Move to x, y, z
Rectangle, Rectangle3	2 and 3 dimensional rectangle

functional capabilities required for a system to be CORE compatible. Levels of capability are specified so that even the low performance graphics systems can achieve a level of CORE compatibility. The CORE definition is detailed in its specification of the functions a CORE compatible system must support. However, it falls far short in defining the interface to physical devices and in discussing data base issues.

Even with its inevitable shortcomings, the CORE Standard has had a tremendous impact on graphics design and development. It was the first serious attempt to consolidate techniques and ideas related to graphic design in such a fashion that it could be used as a standard to guide future designs.

The more recent GKS proposed standard addresses the issue of physical hardware more completely. It provides at least some data structures support by defining the kind of data that must be stored regarding workstations, error conditions, and metafiles. This standard also falls short of tackling the problem of how to adequately define the software for the physical hardware interfaces.

The interface area is at once the most difficult, and possibly the most necessary of all the technical areas to be solved. Because software developers tend not to develop hardware and vice versa, there is a natural gulf between the two due to physical separation. This leads to poor interfacing, or large volumns of specilized software for each software/hardware configuration.

Two very recent efforts offer some hope for the future. One is the definition of the virtual device interface (VDI). This concept works very similarly to the p-system under Pascal. In such a system, a theoretical machine is postulated. All software can be designed to function with the virtual machine that has been defined. Only a small portion must then make the final conversion to the real

device. The second concept has become a kind of standard due to its success. The NAPLPS system, which grew out of the videotext arena, shows promise for standardizing the communications with graphics hardware.

The NAPLPS system is a communication protocol and was originally used to transmit videotext information across telephone lines. The system is not terribly complex, and has embedded within it the capability to define not only text, but symbology as well. All data transmitted is in standard ASCII format.

The beauty of NAPLPS is that it is a concise, hardware independent, expandable system. It is quite efficient from the communications standpoint which makes it attractive for transmitting graphical data at high rates. With proper encoders and decoders, ANY I/O device can use the standard. This format is being considered by the national standards committees in both the USA and Canada. It is also becoming more widely accepted by the practitioners in the field. Most notable proponents are business firms with much to gain in the graphics area but little to gain from videotext. Thus it may become a standard by virtue of its effectivness [Gordon, 1983: 61-66].

The NAPLPS system uses the ASCII character set with International Standards Organization (ISO) code-extensions. This technique expands the 128 ASCII character set so that numerous sets of commands and symbols are available and can be moved into and out of an 'in use' table. This

code-extension technique also permits the addition of user defined sets. Thus the NAPLPS system can be expanded and yet remain completely consistent with its baseline definition.

This protocol appears to have many of the features necessary to support true device independence. Systems using NAPLPS need to have the proper communication links. With that and the protocol, graphics output devices would truly be interchangeable. Time will tell if NAPLPS becomes a major force in the graphics world. As with anything else, vendors are somewhat hesitant to commit hardware and software development to it until the final, approved version is available. But, the promise is there. With proper development, it may fill the gaps in the CORE and GKS standards and help produce a complete system for graphics standards.

Software Techniques to Handle Dialogues

Extensive comments have been made regarding the needs for the 'human' side of the human/machine interface. Appendix B discussed the human factors topics at great length. Within this discussion were identified critical areas to be addressed in the implementation of the DESIGNS dialogue.

The software for the dialogue is based on, and considers the following areas,

(1) the 'natural language' analogy,

(2) the communication of information via the user/machine interface,

(3) the definition of the dialogue as a language,

(4) the techniques available for 'conversation',

(5) the protection from errors during a 'conversation', and

(6) the tools available to support the dialogue design.

Throughout this discussion the terms dialogue and conversation will be used somewhat interchangeably. This stems from the underlying language analogy that likens the interchange between user and machine to an interpersonal conversation. It is this analogy with human to human exchanges that drives designers of human to machine conversations to pattern after the familiar form of the spoken language. It is presumed that this form of exchange is more readily understood to the user. However, it is certainly not readily understandable to the machine!! Artificial intelligence systems deal more heavily with this issue of natural languages. With conventional technology, true natural language exchanges are not practical.

It is reasonable to pattern certain features of the exchange after interpersonal communications. It is then necessary to maintain simple rules and vocabulary for the exchanges, use concepts familiar to the user, and provide for some extensibility [Van Dam, 1982: 218-219].

The actual communication between the user and the system is actually defined on multiple levels. The user works through the provided 'language' to gain a view of the data

stored within the system. The language is the window into capabilities and information contained within the system. But the overall communication is not at all symmetrical due to the vast differences between the methodologies used by human and machine to process information [Martin, 1973: 3-8].

The language is specified in terms of semantic, syntactic, and lexical characteristics. The semantics of the dialogue form used describe the functional capabilities of the system. It specifies what information is necessary to accomplish the function, and what the result is. In addition, it determines which functions are in the perview of the human, and which are those of the machine.

The syntax of the dialogue form characterize the actual exchange between user and system. It defines the 'rules' of the language, specifying legal input and output sequences. This is just as the syntax of any language, spoken or written, defines those constructs which can be correctly and properly used which the language.

The lexical, or procedural, portion of a language definition, specifies how input/output primitives are formed. It comprises the procedural capabilities necessary to actually accomplish an operation [Martin, 1973: 309, Van Dam, 1982: 210-225].

It is important that the integrity of the conversation be maintained as it is developing. When individuals speak, they are capable of filtering out erroneous or unrelated information. At least to some extent, the dialogue between

user and computer must be likewise protected from incorrect data. This is a complicated task and requires some of the most sophisticated processing techniques available.

E. B. James discusses this issue in terms of 'protective' ware. He describes a layer of software that logically resides between the user and the actual communication channel, and filters out undesirable portions of the conversation. This 'protects' the integrity of the dialogue and assures that only valid conversation commands are carried out [James, 1981: 349-350].

'Protective' ware, or bullet proofing of the system software, is largely a function of the dialogue structure selected. In general, computer initiated conversations are easier to protect against errors. When designing the dialogue structure, error detection and/or correction can be taken into account. By considering

(1) the psychology of the dialogue format, the liklihood of errors can be minimized;

(2) the structure of the dialogue, methods of recognizing and trapping errors can be developed within the dialogue framework and the structure can then aid in error correction;

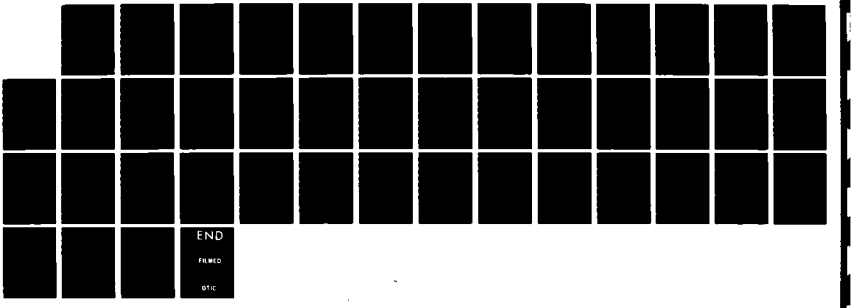
(3) the extent of error propagation, the dialogue must bridge the gap between the real-time conversation and any resultant actions or created files [James, 1981: 450-462].

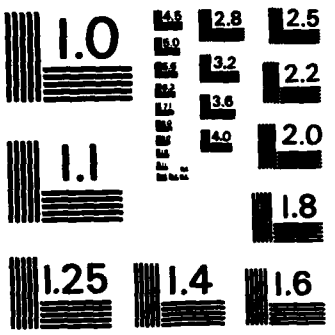
A dialogue developed around these principles provides for a more graceful degradation should a failure occur.

Looking at the time sequence of events in an interactive system's structure is important. State transition diagrams

AD-A159 218 THE INTERACTIVE GENERATION OF ALPHANUMERICS AND SYMBOLOGY WITH DESIGNS ON THE FUTURE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 3/3

UNCLASSIFIED K A ADAMS JUN 85 AFIT/GCS/MA/85J-2 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

explicitly specify the temporal relationships that exist within the structure [Jacob, 1983: 28-34; Kieras and Polson, 1983: 103-106]. Such tools, which are relatively simple to use, can readily define acceptable system states and identify problems within the dialogue structure.

The Impact of Fourth Generation Languages

A discussion of fourth generation languages (4GLs) is relevant to this thesis, because DESIGNS can be considered as a 4GL. The key criterion in defining a 4GL is that the language supports and improves user productivity. The use of the term 'language' is actually a bit of a misnomer, as 4GL systems are not languages in the classical sense [Tufts, 1984; Tharp, 1984: 37]. Indeed, they are support environments for some set of users.

The 4GL systems have evolved from the more traditional languages. They tend to be less procedural than third generation languages such as Pascal or FORTRAN. Individual statements within the 'language' cause more activities to be initiated than in traditional languages. Thus the specification for an activity is more concise and, hopefuly, more natural in form.

The 4GL systems tend to be 'friendly', flexible, and robust. They make efficient use of the user's time, but not necessarily the computer's time. They are almost exclusively on-line, interactive systems [Tufts, 1984]. The development

of such systems has normally been undertaken to solve a specific problem, and to provide a tool for the user. Examples of such systems include various query systems, database systems, report generators, and various graphics output languages.

Within the context of this definition, DESIGNS can be considered as a 4GL. It meets the criterion, and its primary goal is to reduce production time required for cockpit symbology. There are no rigid definitions of a 4GL which preclude identifying the DESIGNS system as an example of fourth generation techniques.

Algorithms for Graphical Tools

Textbooks filled with algorithms for graphical manipulations are commonplace [Foley and Van Dam, 1982; Newman and Sproull, 1979; Pavlidis, 1982]. It is not the focus of this thesis effort either to investigate or develop new algorithmic techniques.

The DESIGNS subset that has been selected for implementation and testing is more functionally oriented. It is a set of tools to manipulate existing symbology to create display formats. Additional capabilities for the DESIGNS environment, that are only postulated at this point, will be more algorith intensive.

Tools which will support the drawing of new symbols from primitives, or support the complex image generation require

graphics algorithms. In places where special graphics algorithms are required, available techniques will be used.

Appendix D. DESIGNS Top-Level SADT Diagrams

The SADT diagrams contained in this appendix illustrate the top-level definition of the DESIGNS system. They follow from the description of functional requirements contained in the fourth chapter of this thesis. Coupled with the pseudocode in the following appendix, they provide the reader with a solid understanding of the project definition.

The SADT diagrams prepared for the DESIGNS definition phase focused on the functional aspect of the system, and looked critically at the activities within that system. The SADT diagrams for the DESIGNS development reflect that point of view.

As a quick review, the following description tells how the SADT diagram is used. The box itself represents the activity of interest. Arrows approaching the box from the left represent input data, arrows approaching from the top represent controlling data, and arrows departing to the right represent output data. The basic concept is that the box activity tranforms the input data to the output data, governed by the control information that constrains the system.

The following SADTs define the important executive levels of the DESIGNS system.

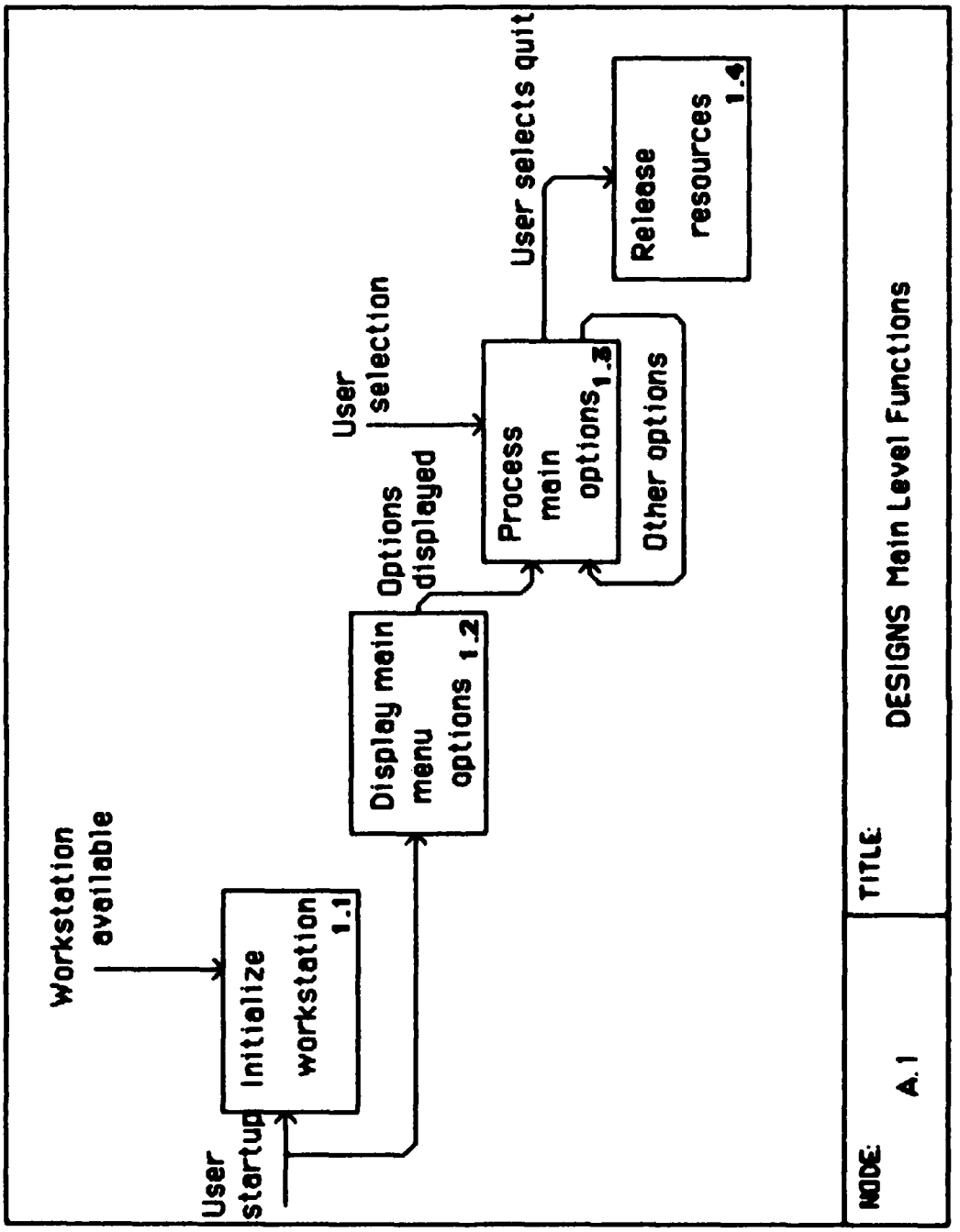


FIGURE 27
Entry Level of DESIGNS - System Startup

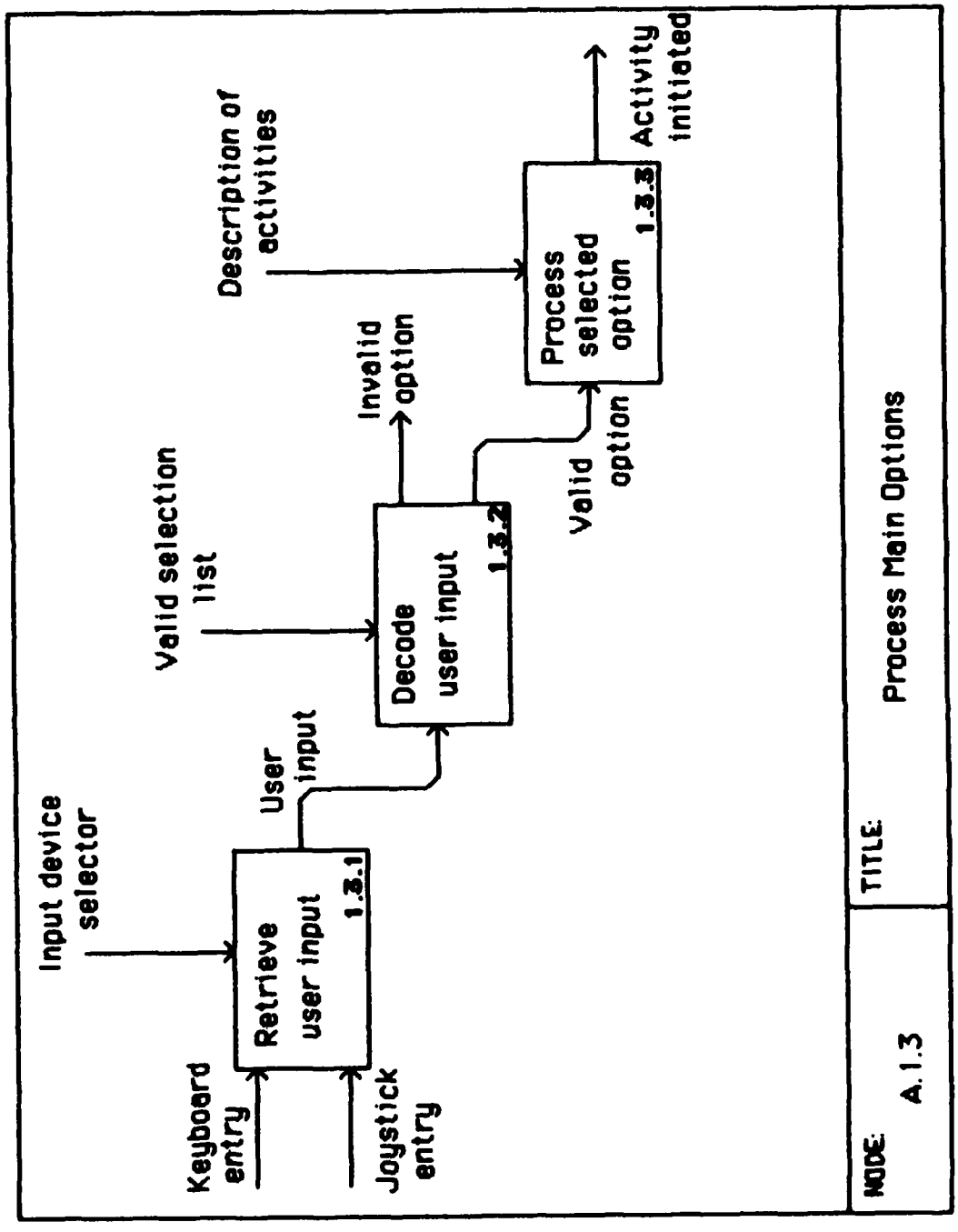


FIGURE 28
DESIGNS Menu Option Processing

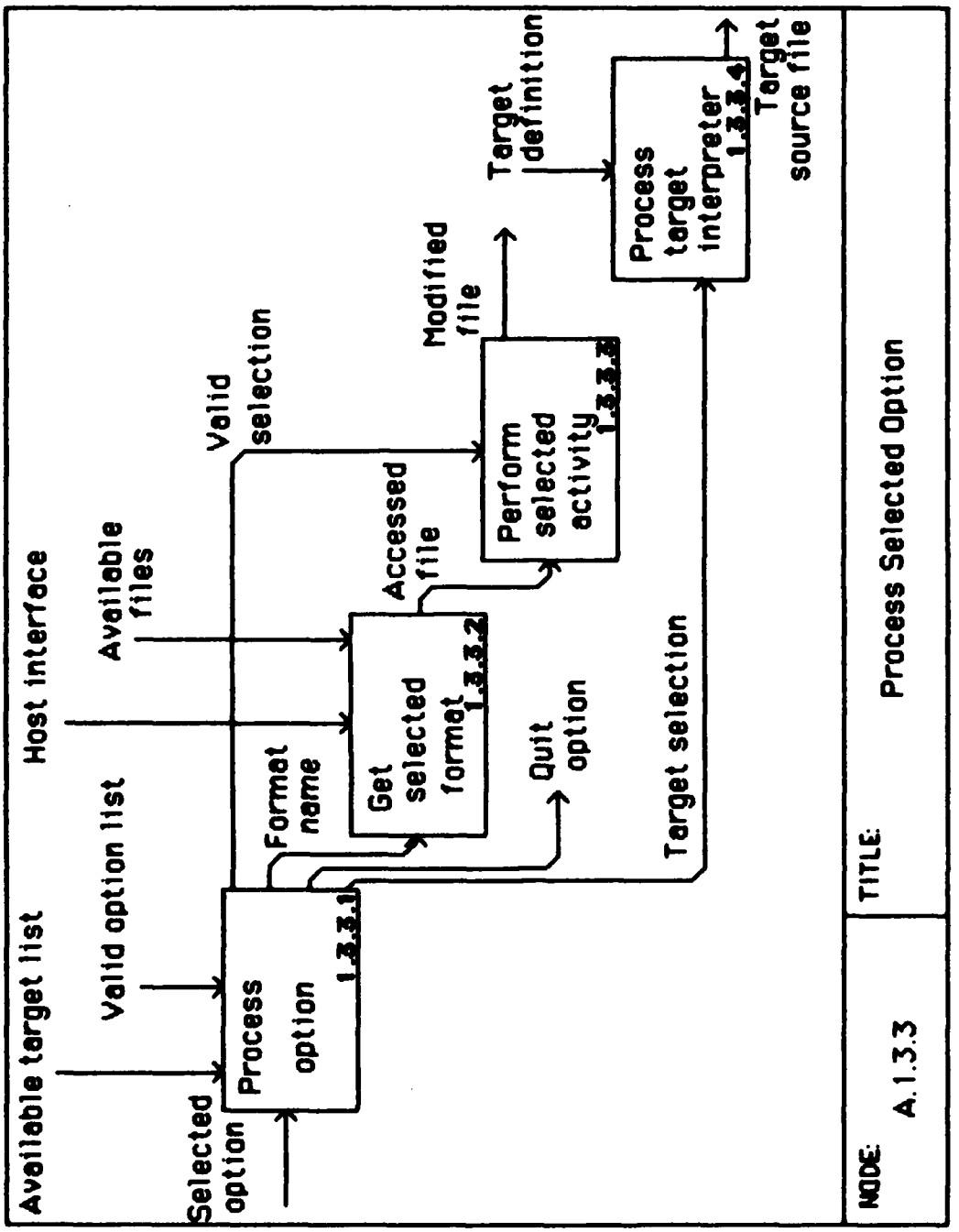


FIGURE 29
DESIGNS Option Selection Processing

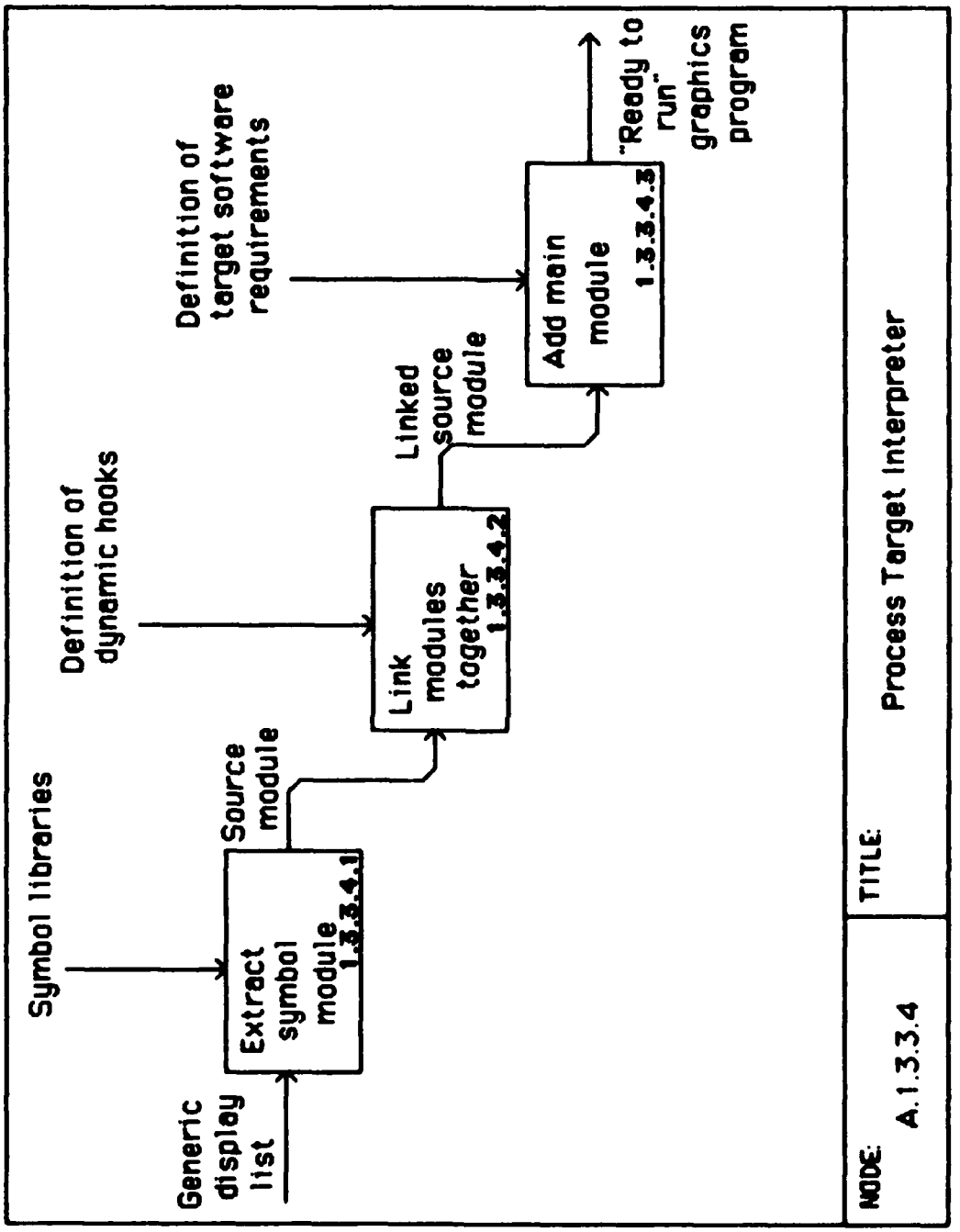


FIGURE 30
DESIGNS Interpreter Activities

Increasing levels of detail exist under each of these SADT charts. This detail has not been included for two main reasons. The first is that some of the details, as discussed in the body of the thesis, have not yet been defined. The second is that the most important information can be desribed without including the more mundane levels of detail.

Appendix E. DESIGNS Pseudocode

This appendix contains pseudocode for the upper-most, functional level of the DESIGNS environment definition. Definitions for every module developed as a part of the thesis effort are not included here in order to present the most informative data in a concise manner. Where appropriate, families of related modules are defined in general terms. This permits the reader to visualize DESIGNS as a complete system, without pondering module specific details.

The software developed for DESIGNS falls into six general categories, which are the:

(1) executive functions of the environment, which 'manage' the entire system and are the routines of primary interest in the pseudocode development,

(2) menu structure and help information, which provide the portion of the user interface which the user actually 'sees',

(3) librarian features which provide all access to the underlying file structures used by the system,

(4) support modules which provide the basic graphics primitives, interface through the host operating system, and other supportive functions,

(5) hardware specific modules which provide the interface both to the system workstation, and the various available graphics devices, and

(6) defined symbol sets that have been developed, and are commonly used within the hosting facility.

The pseudocode contained in this section emphasizes the first three areas. An understanding of the DESIGNS

structure, with its important user interface concepts and librarian features, is necessary. The major functions of the hardware, support, and symbol sets are enumerated. Since the pseudocode for individual modules in these areas tend to be either somewhat repetitive, in the case of symbol sets, or very machine dependent, in the case of hardware and support modules, it is not included here.

The mainline module for DESIGNS performs all of the setup work for the package, invokes the actual DESIGNS executive module, and 'cleans up' after the user has completed the desired job.

The associated data structures for the DESIGNS package are defined in a single block data structure. The description of these structures is contained in the following block of pseudocode.

```
*     PURPOSE of subroutine THESIS

*************************************************************
*                                                           *
*  MAINline program for DESIGNS development environment.   *
*                                                           *
*    D - esign          ! DESIGNS has been developed in     *
*    E - nvironment     ! support of the thesis effort      *
*    S - upporting the  ! required by the Air Force         *
*    I - nteractive     ! Institute of Technology (AFIT)    *
*    G - eneration of   ! to satisfy all the requirements   *
*    N - alpNumerics &  ! for an MS in Computer Systems     *
*    S - ymbology       ! Engineering. ( 1982-1985 )        *
*                                                           *
*************************************************************

*     REQUIRED DATA for subroutine THESIS

*************************************************************
*                                                           *
*    - definition for the different picture segments        *
*      within the DESIGNS systen                            *
*                    : common block - Station_Pictures      *
*                                                           *
*    - definition of basic workstation parameters such      *
*      as intensity levels, character sizing, character     *
*      rotation                                             *
*                    : common block - Control_Information   *
*                                                           *
*************************************************************

*     PSEUDOCODE for subroutine THESIS

*************************************************************
*                                                           *
*    Initialize the workstation hardware                    *
*                                                           *
*    Draw each required DESIGNS help screen                 *
*    Draw the DESIGNS working area - borders, drawing area  *
*    Draw each required DESIGNS menu screen                 *
*                                                           *
*    loop until the user wishes to stop using DESIGNS       *
*         process the DESIGNS options                       *
*    end loop                                               *
*                                                           *
*    Release the workstation hardware for the next job      *
*                                                           *
*************************************************************
```

```
*     PURPOSE of code block BLOCK DATA

*****************************************************************
*                                                               *
*   these common blocks are required for the DESIGNS            *
*   system and are initialized in block data                    *
*                                                               *
*    Device_Control       - specifies workstation options       *
*                           as unit, copy control, on/off,      *
*                           and echoing.                        *
*                                                               *
*    Station_Pictures     - defines picture numbers for         *
*                           each of the separate pictures       *
*                           within DESIGNS                      *
*                                                               *
*    Control_Information - control data for workstation         *
*                           attributes such as intensity,       *
*                           character sizes and rotations,      *
*                           line types, etc.                    *
*                                                               *
*    VG_Definition        - defines the VG graphics             *
*                           characteristics                     *
*                                                               *
*    Symbol_Table         - contains generic definition         *
*                           of the display being created        *
*                                                               *
*****************************************************************

*     PSEUDOCODE for code block BLOCK DATA

*****************************************************************
*                                                               *
*    declare ALL variable types                                 *
*    initialize ALL data values                                 *
*                                                               *
*****************************************************************
```

The executive module manages all of the DESIGNS activities. It interprets the selected options and provides the appropriates response. All error handling and interfaces to the host operating system are through this main executive module. Parallel module exist for the various menu levels.

```
*     PURPOSE of subroutine MAIN_SELECTIONS

*************************************************************
*                                                           *
*   subroutine MAIN_SELECTIONS - executive control for      *
*                        all user interaction with the      *
*                        MAIN menu                          *
*                                                           *
*************************************************************

*     REQUIRED DATA for subroutine MAIN_SELECTIONS

*************************************************************
*                                                           *
*     user selection to stop DESIGNS  :  Quit_Designs       *
*                                                           *
*     workstation definition                        :       *
*                        common block    Device_Control     *
*                                                           *
*     definition of the available picture segments :        *
*                        common block    Station_Pictures   *
*                                                           *
*     attribute information for the workstation    :        *
*                        common block Control_Information   *
*                                                           *
*     the generic DESIGNS display list information :        *
*                        common block    Symbol_Table       *
*                                                           *
*************************************************************

*     PSEUDOCODE for subroutine MAIN_SELECTIONS

*************************************************************
*                                                           *
*     initialize all control variables                      *
*     turn on required pictures to display the MAIN menu    *
*     turn off all unnecessary picture segments             *
*                                                           *
*     loop until user decides to quit the DESIGNS system    *
*                                                           *
```

```
*          read the keyboard/joystick inputs                    *
*          determine which menu option has been selected        *
*                                                               *
*          for the case that the option selected is:            *
*                                                               *
*              create a display -                               *
*                  initialize generic display list              *
*                  create the new display                       *
*                                                               *
*              modify an existing display -                     *
*                  determine which format to be modified        *
*                  display that format on the workstation       *
*                  modify the display                           *
*                                                               *
*              store the current display -                      *
*                  determine the name for the display           *
*                  store the display list                       *
*                                                               *
*              document work -                                  *
*                  determine which format to be documented      *
*                  document the work                            *
*                                                               *
*              assign attributes -                              *
*                  determine which symbol(s)/alpha(s)           *
*                    to have dynamic hooks                      *
*                  establish dynamic hooks in display list      *
*                    according to the user instructions         *
*                                                               *
*              select a target device -                         *
*                  determine which device the user wishes       *
*                    to use                                     *
*                  interpret generic display list for the       *
*                    specified target                           *
*                                                               *
*              dynamic testing -                                *
*                  test the current display with                *
*                    representative inputs signals              *
*                                                               *
*              help -                                           *
*                  provide helpful information specific         *
*                    to the current DESIGNS situation           *
*                                                               *
*              quit -                                           *
*                  stop the DESIGNS system                      *
*                                                               *
*              any invalid option -                             *
*                  notify that selection is invalid, try        *
*                  again                                        *
*                                                               *
*          endcase                                              *
*      endloop                                                  *
*                                                               *
*****************************************************************
```

```
*     PURPOSE of subroutine SYMBOL_SELECTIONS

*****************************************************
*                                                   *
*   subroutine SYMBOL_SELECTIONS - executive control for *
*                                  all user interaction  *
*                                  with SYMBOLIC menus   *
*                                                   *
*****************************************************

*     REQUIRED DATA for subroutine SYMBOL_SELECTIONS

*****************************************************
*                                                   *
*   user selection to stop SYMBOLIC operations   :  *
*                                     Quit_Symbol   *
*   attribute information for the workstation    :  *
*                        common block  Control_Information *
*                                                   *
*   the generic DESIGNS display list information :  *
*                        common block  Symbol_Table *
*                                                   *
*****************************************************

*     PSEUDOCODE for subroutine SYMBOL_SELECTIONS

*****************************************************
*                                                   *
*   initialize all control variables                *
*   turn on required pictures to display SYMBOLIC menus  *
*   turn off all unnecessary picture segments       *
*                                                   *
*   loop until the user quits the SYMBOL operations *
*                                                   *
*       read the keyboard/joystick inputs           *
*       determine which menu option has been selected  *
*                                                   *
*       for the case that the option selected is:   *
*                                                   *
*           any valid symbol -                      *
*             establish default scaling for display list *
*               symbol                              *
*             draw the symbol in workstation field of  *
*               view                                *
*             update generic display list to include new *
*               symbol                              *
*                                                   *
*           help -                                  *
*             provide information specific to current  *
*               DESIGNS situation                   *
```

```
*                                                                     *
*           quit -                                                    *
*             stop the DESIGNS system                                 *
*                                                                     *
*           any invalid option -                                      *
*             notify user selection is invalid, try again             *
*                                                                     *
*       endcase                                                       *
*                                                                     *
*   endloop                                                           *
*                                                                     *
***********************************************************************
```

Each unique graphics target has its own interpretive section within DESIGNS. While each of these interpreters is specific to the hardware and software that make upthat system, the PACER/VG interpreter is 'typical' of the construcion of such a module.

```
*     PURPOSE of subroutine PACER_VG_INTERPRETER

***********************************************************************
*                                                                     *
*   subroutine PACER_VG_INTERPRETER - mainline module of              *
*                          interpretive package for the               *
*                          PACER/VG graphics system.                  *
*                                                                     *
***********************************************************************

*     REQUIRED DATA for subroutine PACER_VG_INTERPRETER

***********************************************************************
*                                                                     *
*   definition of the VG graphics system : common block               *
*                                          VG_Definition              *
*                                                                     *
*   definition of generic display list created during a               *
*     DESIGNS session          : common block Symbol_Table            *
*                                                                     *
***********************************************************************

*     PSEUDOCODE for subroutine PACER_VG_INTERPRETER

***********************************************************************
*                                                                     *
*   loop for the number of modules to be retreived                    *
*                                                                     *
*       set VG symbol prefix, based on generic DESIGNS                *
*           display list contents                                     *
```

```
*                                                              *
*   endloop                                                    *
*                                                              *
*   generate complete name list for resultant VG source        *
*                                                              *
*   open the required VG symbol libraries                      *
*                                                              *
*   get necessary static/dynamic modules from libraries        *
*                                                              *
*   close all libraries                                        *
*                                                              *
****************************************************************
```

The library support features include the following capabilities:

(1) error handling,

(2) opening and closing of the necessary library files,

(3) reading and writing files,

(4) searching for files,

(5) modifying files to create the required dynamic links, and

(6) managing the namelist of required symbols associated with the library.

The general support functions include:

(1) basic graphics primitives such as drawing circle, arcs, and short vectors,

(2) error handling through the host interface, and

(3) any general support tool required, such as menu generation.

The hardware support functions referenced here primarily support the DESIGNS workstation. Other hardware specific modules are considered part of the target interfaces. The functions supported in the workstation to date are:

(1) station initialization,

(2) picture segment initialization,

(3) joystick, keyboard, and digitizing tablet interface,

(4) selective turing on/off of pictures,

(5) 'picking' of picture segments, and

(6) de-allocation of the workstation and its resources.

The basic symbol set on which DESIGNS was based is common to a wide variety of HUD applications. It included:

(1) pitch ladder,

(2) fixed reticle and aiming reticle with range bar,

(3) flight path marker,

(4) Instrument Landing System (ILS) bars,

(5) horizon line, and water line markers,

(5) roll indicators,

(6) various forms of scales, and

(7) various commanded path and target information.

Typical of the structural definition of a symbol is the following definition of the pitch ladder symbology.

```
*     PURPOSE of subroutine PITCH_LADDER
************************************************************
*                                                          *
*   subroutine PITCH_LADDER - part of workstation symbol   *
*                             set                          *
*   Draws standard symbolic representation of aircraft     *
*     pitch ladder.                                        *
*                                                          *
************************************************************
```

```
*     REQUIRED DATA for subroutine PITCH_LADDER

*****************************************************************
*                                                               *
*   x,y coordinates for the symbol center : X_Center,           *
*                                           Y_Center            *
*   width of the ladder symbol            : Ladder_Width        *
*   height of the ladder symbol           : Ladder_Heigh        *
*   symbol attributes                     : Intensity,          *
*                                           Scaling             *
*                                                               *
*****************************************************************

*     PSEUDOCODE for subroutine PITCH_LADDER

*****************************************************************
*                                                               *
*   set the default values for the symbol attributes            *
*                                                               *
*   map x,y coordinates into workstation viewing space          *
*                                                               *
*   draw the symbol as illustrated:                             *
*                                                               *
*          10  ____________________________  10                 *
*                                                               *
*                                                               *
*                                                               *
*           5  ____________________________   5                 *
*                                                               *
*                                                               *
*                                                               *
*           0  ____________________________   0                 *
*                                                               *
*                                                               *
*                                                               *
*          -5  - - - - - - - - - - - - - -   -5                 *
*                                                               *
*                                                               *
*                                                               *
*         -10  - - - - - - - - - - - - - -  -10                 *
*                                                               *
*****************************************************************
```

Appendix F. DESIGNS User Guide

This user's guide is meant to assist the beginning user of the DESIGNS environment. It is not, as of this printing, complete since the facility itself is not yet completed. It does, however, document those capabilities that currently function.

The DESIGNS package resides in the Flight Dynamics Laboratory's simulation facility that is housed within the Control Synthesis Branch in Building 145, Area B, Wright Patterson AFB, Ohio. It is hosted on the SEL 32/2750 computer that has been designated as the graphics front-end processor.

In order to use the DESIGNS system, the user must first log onto the SEL computer. This is accomplished by logging in under the username 'graphics'. There is no key associated with the name, so enter a carriage return when a key is requested.

The logon procedure automatically places the user in the correct directory to access all available graphics tools, including DESIGNS. To activate the desired tool, in this case DESIGNS, enter 'displays' (without the single quotes and in either upper or lower case) when the standard operating system prompt is visible.

The prompt line will look like this TSM°displays

Starting this macro invokes the actual graphics executive program. The executive displays a menu of available graphics systems. To use DESIGNS, select the Megatek B option, which is option 2. The next menu level that appears contains the list of available software packages for that particular piece of hardware. This menu level is set up as a series of pages. As of this time, DESIGNS is listed on the second page. Therefore to select it, the user must first go to the next page by selecting 'N' for Next. The next page of programs is listed and DESIGNS appears as number 6. Select 6, and DESIGNS will be automatically started.

It is normal practice at this point to stop the graphics executive program unless other programs are to be started on other graphics machines. Therefore select quit from this menu level which returns the user to the initial hardware list menu. From here option 0 stops the graphics executive. This option is taken in order to release the computer terminal for other use. Since all dialogue between the user and the DESIGNS system occurs via the Megatek unit itself, the computer terminal is not used therefore should be released.

DESIGNS starts by displaying a general 'help' screen in order to orient the user to the basic forms of communicating with the system. Once the user has read this, press the carriage return key on the Megatek keyboard to continue. A short period of time will elapse while the DESIGNS data

structures are initialized. When this is complete, the main menu will appear on the screen.

The main DESIGNS menu, which has been illustrated in Figure 21, contains all of the top-level functions for manipulating the generic display list. Selections are made from this, or any other, menu by one of two methods. The first method involves typing in the command from the keyboard. On the main menu, the command takes the form of the first word in the menu option. For example, select 'Create Format' by typing in 'create' followed by a carriage return.

The second method for selecting an option is to position the cursor by using the Megatek joystick near the option to be selected. Once the cursor is positioned, push the button on top of the joystick to select the option.

In either case, if the typed command is incorrect or the joystick button is pushed while not near an option, an error message will be displayed asking the user to re-select. Once a valid command is selected, the function to be performed is initiated.

Throughout the DESIGNS system, on-line help can be accessed by typing a question mark. Information is provided relevant to the current state of the system. Once the user has completed using the help information, typing a carriage return restarts the DESIGNS system at the point at which help was requested.

To exit any menu, and return to the menu immediately preceeding it, the user need only type 'quit'. This always returns to the calling menu. To stop the DESIGNS system entirely, the user types 'quit' from the main menu level. At the conclusion of a session the user-created files are all saved for use during a later session.

Appendix G. Requirements for GRAFEXEC

I. Introduction

A. Numerous independent graphics packages currently hosted on the SEL 32/2750

B. SEL 2750 to be the host for future graphics devices

C. Preliminary software testing for an executive exists - MEGOPTSx

D. Addresses training needs by minimizing req's for detailed graphics knowledge by most users

E. Another step in the development of an integrated graphics environment

II. Purpose of Executive

A. Provide a consistent interface to the various available packages

B. Provide extensibility as new packages/hardware are added to the SEL system

C. Reduce training requirements for end users

III. Functional Requirements of the Graphics Executive

A. Must appear to the user as a single through which other available options are selected

B. Must be capable of selecting options and initiating their execution in some fashion, as well as terminating their execution in some fashion

C. Must be able to allocate the target device to be used for the display

D. Must recognize if the designated target is currently allocated and require confirmation to terminate the previous graphics package and reallocate the device

E. Must release whatever resources it uses for its physical presentation of the user interface when finished, so that they are free for other tasks

F. Must initialize the target device (clear the screen) and deallocate devices when a graphics task is terminated

G. Should provide its own error handling, and not confront the user with the potential of falling into unclear system error messages

IV. Functional Requirements of the Support Graphics Packages

A. Define a common interface with the executive for control, data passing, etc.

B. Should not provide an internal mechanism for terminating its own execution is this prevents the executive from knowledge that the task is gone and thus devices deallocated

C. Should conform to the standard data structure established for simulation data passing if it is to be driven by a simulation

D. Should not assume any of the specified executive roles

V. Implementation Requirements for the Software Developed

A. Must be compatible with the SEL 32/2750 host

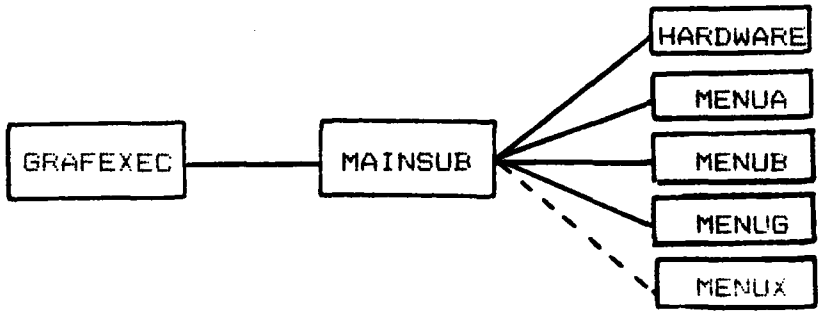
B. Design and implementation should permit unlimited addition of new capabilities (application programs) within the existing framework of the executive design. Additional applications can be made available as options without regard to memory constraints although there will be some real limit, both memory and time, as to how many can execute simultaneously.

C. Must be compatible with shared memory configuration if it is to communicate with other mainframe software.

GRAPHICS EXECUTIVE DOCUMENTATION

1.0 Introduction - This document describes the graphics executive which interfaces graphics software packages on the SEL 32/27 to target hardware devices. The executive enables a user to activate any task on any device through a single program. It also provides the capability to abort tasks if the desired device has previously been allocated or the user wishes to change the tasks which are running.

2.0 Display Routine Structure -



The purpose of GRAFEXEC, the main program, is to call MAINSUB, the highest level subroutine.

2.1 Subroutines - This section describes the function of each of the GRAFEXEC subroutines.

2.1.1 Procedure - MAINSUB - This procedure calls the menu display subroutines and accepts the user's input for software, hardware, and menu control. It then uses that information to activate the chosen task, or to change the page of the menu to be displayed. It also reads in a file which contains current information about the active tasks. In the event that the device chosen is unavailable, due to previous allocation, this information is provided to the user.

2.1.1.1 Inputs - CHOICE - user selections

2.1.1.2 Outputs - PAGENO - an integer specifying which page of the software menu to display. It is set to one before it is passed to the menu subroutines.

2.1.2 Procedure - MENUA - This procedure displays the software options for MEGATEK A. It also provides options to control which page of the menu to display. At present, there are five pages and a capability for twenty options.

2.1.2.1 Inputs - PAGENO - an integer specifying which page of the menu to display.

2.1.2.2 Outputs - None

2.1.3 Procedure - MENUB - Same as above for MEGATEK B

2.1.3.1 Inputs - Same as above

2.1.3.2 Outputs - Same as above

2.1.4 Procedure - MENUG - Same as above for GAERTNER. At present, however, no options exist.

2.1.4.1 Inputs - Same as above

2.1.4.2 Outputs - Same as above

2.1.5 Procedure - MENUX - Additional menus of the same format as MENUA, etc., may be incorporated to accomodate the addition of new hardware devices.

2.1.5.1 Inputs - as needed

2.1.5.2 Outputs - as needed

2.1.6 Procedure - HARDWARE - This procedure displays the graphics hardware options. It then accepts the user s choice and attempts to allocate the device. If this attempt fails, the user is given options to abort the test,

select another device, or quit. If necessary, additional pages may be added in the same fashion as MENU).

3.0 Expansion - This section deals with the steps necessary to expand either the hardware or software menus.

3.1 Existing Software Menus - Adding additional options or pages. All of the software menus have the same format, therefore these instructions apply to each of them.

3.1.1 Modifications to MAINSUB - When a page is added to a menu, then the index of the array ENTRIES(X) should be incremented by 5. The index should be equal to the maximum number of options in any menu, plus four for the page control entries. For example, if MENUA had 5 pages, and MENUB had 6 pages, then ENTRIES(X) would not have to be expanded if another page were added to MENUA, because the array is already large enough to satisfy 6 pages. If, however, a seventh page were added to either menu, then the array would have to be expanded. Also, the data for ENTRIES should be modified to include the numbers of the new options.

The integer array NUMBOPTS(X) should be expanded to include the new number of options. NUMBOPTS(1) corresponds to NEGATER A, so it should be set equal to the total number of used options in MENUA. Even though 24 options total are available, the number of options actually used may range from 1-14. When NUMBOPTS is used, it is indexed by DEVICE. For an explanation of the relationship between DEVICE and the hardware device see section 3.1.3.

Example Select Case Statement:

```
SELECT (VOL 1
  CASE 2
    IF (DEVICE.EQ.1) THEN
      NAME ='TRASHA'
    ELSEIF (DEVICE.EQ.2) THEN
      NAME='TRASHB'
    ENDIF
    ACTIVE(DEVICE)=NAME
    WRITE(7,20,ERR=60,IOSTAT=ISTATW)(ACTIVE(I),I=1,5)
    CALL M:ACTIV(INAME,IERRSTAT,$30)
    CALL HARDWARE(DEVICE,ACTIVE)
END SELECT
```

NOTE: NAME is a CHAR*8 variable, INAME an INT*8,the two are equivalent. The M:ACTIV call requires an integer doubleword so INAME is used.

The new option(s) should be added to the SELECT CASE I. The number of the case will be equal to the number of the option plus four, for page control. For example, option '12' would be 'CASE 16', etc. The integer variable 'NAME' should be set equal to the program name: NAME = 'TESTXX'. The M:ACTIV call will remain the same, as will the statement ACTIVE(DEVICE)=NAME and the WRITE statement. If, however, the program will run on more than one device then an

IF-THEN-ELSE statement of the form shown must be included for the appropriate devices. The relationship between the values of DEVICE and the devices will be discussed in the HARDWARE section.

The value of LASTPAGE(X) must be set equal to the number of the last page number for that particular menu. LASTPAGE(1) corresponds to MENUA, LASTPAGE(2) to MENUB, etc. When the array is used, it is indexed by DEVICE. For an explanation of the values of DEVICE, see section 3.1.3

3.1.2 Modifications to MENU(X) - Any additional pages should be created with the same format as those already existing. The new options should be added next to the appropriate option number, to coincide with the SELECT CASE statement in MAINSUB (option 'X' to CASE 'X+4').

The array MESS (X,21) must be expanded so that X equals the number of pages in the menu. Also, the added data statement for MESS should be of the form DATA (MESS (pageno, J), J=1,21) followed by the page format.

3.1.3 Modifications to HARDWARE - No changes need to be made to HARDWARE in order to change the software menu. However, some information should be provided as to the role of ACTIVE (DEVICE). DEVICE is an integer variable whose value corresponds with options in the hardware menu. For example, 1 corresponds to MEGATEK A, '2' to MEGATEK B, etc. ACTIVE is an array which contains the names of active programs. ACTIVE(1) contains the name of the program running on Megatek A, ACTIVE(2) the name of the program on Megatek B, and so on.

3.2 Adding Software Menus - Software menus should be added to correspond with new hardware devices.

3.2.1 Modifications to MAINSUB - The initial software menu call must be modified to include the new menu.

Example:

```
IF (DEVICE.EQ.1) THEN ! MEGATEK A
  CALL MENUA (PAGENO)
ELSEIF (DEVICE.EQ.2) THEN ! MEGATEK B
  CALL MENUB (PAGENO)
        .
        .
        .

ELSEIF (DEVICE.EQ.5) THEN ! NEW DEVICE
  CALL MENUX (PAGENO) !NEW MENU
ENDIF
```

The array LASTPAGE must be expanded to include the last page number of MENUX.

If the existing software can run on the new hardware, or new software can run on existing devices, then the IF-THEN-ELSE code in the SELECT CASE statement should be modified.

Example for existing case:

```
CASE 5
IF (DEVICE.EQ.1) THEN  ! MEGATEK A
  NAME = 'FORTYA'
ELSEIF (DEVICE.EQ.2) THEN  ! MEGATEK B
  NAME ='FORTYB'
ELSEIF (DEVICE.EQ.5) THEN  ! NEW DEVICE
  NAME ='FORTYX' ! VERSION OF FORTY FOR NEW DEVICE
END IF
```

Example for new case:

```
CASE X
IF (DEVICE.EQ.1) THEN
  NAME = 'TASKA'          !NEW SOFTWARE FOR MEGATEK A
ELSEIF (DEVICE.EQ.5) THEN
  NAME = 'TASKX'
ENDIF
```

3.2.1.1 Inputs - CHOICE, user input

3.2.1.2 Outputs - New LASTPAGE value for new menu.

3.2.2 Modifications to MENUX - The new menu can be copied almost exactly from those existing. The only change would be the name, and the available options.

3.2.2.1 Inputs - PAGENO value

3.2.2.2 Outputs - None

3.2.3 Modifications to Hardware - These will be discussed under the heading of Adding New Hardware Options. The user should keep in mind, however, the correspondence between the value of DEVICE and the device chosen, as this is used in IF-THEN-ELSE statements throughout procedure MAINSUB.

3.2.3.1 Inputs - User choices DEVICE, task names in ACTIVE

3.2.3.2 OUTPUTS - DEVICE

3.3 Adding New Hardware Options - As new graphics hardware devices are added to the system, they and their associated software may be incorporated into the executive.

3.3.1 Modifications to MAINSUB - The initial call to the software menu and the SELECT CASE statement should be modified as discussed in section 3.2.1 under Adding Software Menus.

If mutiple pages are added to the HARDWARE menu then array ACTIVE should be expanded to include the new options.

3.3.1.1 Inputs - CHOICE - user selections

3.3.1.2 Outputs - PAGENO - page of menu to be displayed

3.3.2 Modifications to MENUX - None, except for the addition of a menu to correspond with the new device.

3.3.2.1 Inputs - PAGENO value

3.3.2.2 Outputs - None

3.3.3 Modifications to HARDWARE - Add new option to hardware menu.

Add new option to SELECT CASE statement. The only change will be the device number in the OPEN statement.

Example:

```
 OPEN (UNIT=9, DEVICE=device number, IOSTAT=ISTAT,
+       WAIT=.FALSE., ERR=99)
```

If the status of the device (available or unavailable) cannot be determined with an OPEN statement then another method must be implemented.

Implement a method to clear the device's screen.

Example:

```
IF (DEVICE.EQ.1) THEN  ! CLEARS MEGATEK A
  CALL M:ACTIV('CLRMEGA', ERRSTAT, $30)
ELSEIF (DEVICE.EQ.2) THEN  ! CLEARS MEG B
  CALL M:ACTIV('CLRMEGB', ERRSTAT, $30)
ELSEIF (DEVICE.EQ.3) THEN  ! CLEAR NEW DEVICE
  CALL M:ACTIV ('CLRNEW', ERRSTAT, $30)
ENDIF
```

If new pages are added to the menu then arrays ACTIVE, TEMPACTIVE, and INAME must be expanded.

3.3.3.1 Inputs - ACTIVE - array which contains names of active programs.

DEVICE - index for ACTIVE, corresponds with device numbers in menu.

3.3.3.2 Outputs - DEVICE - integer, used in MAINSUB to activate programs on correct device.

4.0 Variables - This section describes the purpose of each variable in the executive.

4.1 Procedure - MAINSUB

LOGICAL NOTFOUND - Controls the DO WHILE (NOTFOUND) loop. It is set to .TRUE. initially. When a proper choice is made from the software menus it is set to .FALSE. and program execution continues.

INTEGER*1 PAGENO - Represents the page of the chosen software menu to display. It is initialized to '1' when the program is first activated, then it is controlled by user selections from the software menus (NEXT PAGE, PREVIOUS PAGE, BEGINNING).

INTEGER*1 LASTPAGE(5) - This array contains the numbers of the last pages of each menu. It is used to provide a 'wrap-around' capability for menu control. For example, if PAGENO=1 and PREVIOUS PAGE is selected, then the menu 'wraps around' to the last page.

INTEGER*1 NUMBOPTS(5) - This array contains the number of options in each menu. It is used in the DO WHILE (NOTFOUND) loop which determines if a user selection is invalid.

INTEGER*1 DEVICE - Passed in from HARDWARE. It is used to index arrays. Each value corresponds with a hardware device - '1' for MEGATEK A, '2' for MEGATEK B,etc.

INTEGER*8 INAME - Used in M:ACTIV call, which requires an integer doubleword. Equivalenced to CHARACTER*8 NAME, which represents the task name in other cases.

CHARACTER*2 CHOICE - Represents the user's choice from the software menus.

CHARACTRER*8 ENTRIES(24) - An array containing all possible inputs for CHOICE.

CHARACTER*8 NAME - Represents the name of the file to be activated. Equivalenced to INTEGER*8 INAME.

CHARACTER*8 ACTIVE(6) - An array which contains the names of the active tasks.

CHARACTER*26 ENTER - Types the message ENTER SELECTION>> to prompt the user.

4.2 Procedure - MENUA

INTEGER*1 PAGENO - Represents the page number of the menu to display.

CHARACTER*60 MESS(5,21) - Contains the format for the menu. Each page is 21 lines long and 60 characters wide.

4.3 Procedure - MENUB

INTEGER*1 PAGENO - Same as above.

CHARACTER*60 MESS(5,21) - Same as above.

4.4 Procedure - MENUG

INTEGER*1 PAGENO - Same as above.

CHARACTER*60 MESS(5,21) - Same as above.

4.5 Procedure - HARDWARE

CHARACTER*70 OPTS(23) - Contains the format for the hardware menu. The page is 23 lines long and 70 characters wide.

CHARACTER*70 BUSYOPTS - Contains the format for the menu when a device is busy. It is 15 lines long and 70 characters wide.

CHARACTER*6 ACTIVE(6) - An array which contains the names of the active tasks.

CHARACTER*8 TEMPACTIVE(6) - Has the same contents as ACTIVE(6). It is used because input parameters (ACTIVE) to a subroutine cannot be equivalenced.

INTEGER*1 DEVICE - Represents the user's hardware selection. It is used throughout the program as an array index.

INTEGER*8 INAME(6) - Represents the task names. It is used to abort the active tasks.

CHARACTER*26 ENTER - Types the message ENTER SELECTION>> to prompt the user.

LOGICAL OK - Used in DO WHILE (OK) loop. It is set to .TRUE. before entering, and then to .FALSE. when a valid user selection is made.

5.0 Source for Executive - Listed below are the names of the libraries and compilation macros which are used in conjunction with the GRAFEXEC. They are located in directory EXEC.

5.1 GRAFEXEC - Source code is in library EXEC. To compile use macro COMPEXEC.

5.2 FORTYX - The source code is identical for FORTYA and FORTYB and is contained in library PARM. To compile use either COMPAMEG or COMPBMEG. For an explanation of these see section 5.7.

5.3 TRACKINX - The source code is identical for TRACKINA and TRACKINB and is contained in library TRAK. To compile use either COMPAMEG or COMPBMEG. For an explanation of these see section 5.7.

5.4 TERRAINX - The source code is identical for TERRAINA

and TERRAINB and is contained in library TERR. To compile use either COMPAMEG or COMPBMEG. For an explanation of these see section 5.7.

5.5 DATATAB - The source for DATATAB is contained in library TLIB. To compile use COMPDTAB.

5.6 MPD - The source for MPD is contained in library MPD. To compile use COMPMPD.

5.7 COMPAMEG and COMPBMEG - To compile any of the parameter displays for MEGATEK A use COMPAMEG with the following format:

COMPAMEG taskname library

For example, to compile FORTYA enter from TSM:

COMPAMEG FORTYA PARM

To compile any of the parameter displays for MEGATEK B use COMPBMEG with the same format. For example, to compile TRACKINB enter from TSM:

COMPBMEG TRACKINB TRAK

VITA

Ms. Karyl A. Adams was born 8 January 1952 in Urbana, Ohio. She was graduated as the salutatorian of the class of 1970 from Urbana High School. She recieved the degree of Bachelor of Science in Electrical Engineering from Ohio Northern University in May 1975. Following graduation, Ms. Adams was selected as an Olin Fellow at Washington University in St. Louis, Missouri and was enrolled in the graduate biomedical engineering program. She became a civil service employee in January 1976 with the Air Force Flight Dynamics Laboratory at Wright Patterson AFB, Ohio. While with the laboratory, she has been responsible for numerous software development efforts in support of research simulation programs. She has been project leader for the Total Aircrew Workload Study (TAWS), the KC-135 Tanker Avionics and Aircrew Complement Evaluation (TAACE), and the X-29 demonstrator program. In her current position, Ms. Adams is team leader for a group of engineers designing and implementing graphics software in support of the entire simulation facility. She completed the master's program as a part-time student at AFIT while continuing her work within the laboratory.

Ms. Adams was selected to the national women's scholastic honorary, Mortarboard, in 1975. She was selected as the outstanding Scientist and Engineer within the Flight Dynamics Laboratory in 1978. She is listed in Outstanding

Young Women of America, the premier edition of Who's Who in Frontier Science and Engineering, 1984/85, and the International Who's Who in Engineering, 1985.

Permanent Address: 7578 Mt Hood
Huber Heights, Ohio 45424

UNCLASSIFIED

CURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS:
2a. SECURITY CLASSIFICATION AUTHORITY:	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE:	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GCS/MA/85J-2	5. MONITORING ORGANIZATION REPORT NUMBER(S):
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENC	7a. NAME OF MONITORING ORGANIZATION:
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code):
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Control Synthesis Branch; 8b. OFFICE SYMBOL (If applicable): AFWAL/FIGD	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:
8c. ADDRESS (City, State and ZIP Code): Air Force Wright Aeronautical Labs, Wright Patterson AFB, Ohio 45433	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): see Box 19	
12. PERSONAL AUTHOR(S): Karyl A. Adams	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM ____ TO ____; 14. DATE OF REPORT (Yr., Mo., Day): 1985 June; 15. PAGE COUNT: 217	
16. SUPPLEMENTARY NOTATION:	
17. COSATI CODES — FIELD / GROUP / SUB. GR.:	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Graphics, Support Environments, Automated Software, Graphics Design Environment, Real-time Simulation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: A DISPLAY ENVIRONMENT SUPPORTING THE INTERACTIVE GENERATION OF ALPHANUMERICS AND SYMBOLOGY WITH DESIGNS ON THE FUTURE

Thesis Chairman: Charles Richard

Approved for public release: IAW AFR 190-1.
12 AUG 85
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Charles Richard, Associate Professor	22b TELEPHONE NUMBER (Include Area Code): 513-255-3098; 22c OFFICE SYMBOL: AFIT/ENC

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

Abstract

This development effort investigated the available methods for implementing human-computer interfaces using sophisticated graphics systems, with the goal of designing and implementing an advanced graphics development environment. Such a developmental laboratory is necessary to support current and futuristic crew station design for display-oriented cockpits. The result of theis effort was the development system - DESIGNS.

The DESIGNS software system was tailored specifically for real-time, pilot-in-the-loop, research and development simulation. Critical features of such a system include fast turn-around time, and the capability to expand as enhanced tools are developed.

The resulting system supports interactive development of heads-up and heads-down display symbology. It generates correct display formatting information for multiple target graphics devices. Future enhancements include rehosting to a more powerful workstation, preferably in Ada, and continued addition of graphics development tools supporting simulation needs.

END

FILMED

11-85

DTIC